March 2019

Modeling Crowd Feedback in the Mobile App Market

Grant S. Williams
Louisiana State University and Agricultural and Mechanical College, grantwill74@gmail.com

MODELING CROWD FEEDBACK IN THE MOBILE APP MARKET

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Grant S. Williams
B.S., McNeese State University, 2012
M.S., McNeese State University, 2014
May 2019

This dissertation is dedicated to my parents, Sandra and Philip. Without their loving support this Ph.D. would not have been possible.

Acknowledgments

I would like to thank my committee members, Dr. Doris Carver and Dr. Gerald Baumgartner, for their time and support to this work. I would also like to thank Dr. Edward Watson, for agreeing to serve as dean's representative on my dissertation committee. A special thanks goes to my Committee Chairman and Advisor Dr. Nash Mahmoud for his enormous support over the past four years. Without his mentorship and expertise, none of this work would have been possible.

Table of Contents

Abstract

Mobile application (app) stores, such as Google Play and the Apple App Store, have recently emerged as a new model of online distribution platform. These stores have expanded in size in the past five years to host millions of apps, offering end-users of mobile software virtually unlimited options to choose from. In such a competitive market, no app is too big to fail. In fact, recent evidence has shown that most apps lose their users within the first 90 days after initial release. Therefore, app developers have to remain up-to-date with their end-users' needs in order to survive. Staying close to the user not only minimizes the risk of failure, but also serves as a key factor in achieving market competitiveness as well as managing and sustaining innovation. However, establishing effective communication channels with app users can be a very challenging and demanding process. Specifically, users' needs are often tacit, embedded in the complex interplay between the user, system, and market components of the mobile app ecosystem. Furthermore, such needs are scattered over multiple channels of feedback, such as app store reviews and social media platforms. To address these challenges, in this dissertation, we incorporate methods of requirements modeling, data mining, domain engineering, and market analysis to develop a novel set of algorithms and tools for automatically classifying, synthesizing, and modeling the crowd's feedback in the mobile app market. Our analysis includes a set of empirical investigations and case studies, utilizing multiple large-scale datasets of mobile user data, in order to devise, calibrate, and validate our algorithms and tools. The main objective is to introduce a new form of crowd-driven software models that can be used by app developers to effectively identify and prioritize their end-users' concerns, develop apps to meet these concerns, and uncover optimized pathways of survival in the mobile app ecosystem.

Chapter 1. Introduction

Mobile application (app) stores have significantly expanded over the past decade to meet the growing demands of the mobile app market [133]. By the end of 2018, the Apple App Store alone is expected to host nearly two million active apps, growing by almost 2,000 apps per day. In terms of economic impact, in 2015, global mobile app revenues amounted to 69.7 billion U.S. dollars, and by 2020, the mobile app market is projected to generate close to 188.9 billion U.S. dollars [4].

After they are published, apps enter a long phase of feature optimization in order to maintain market viability [33, 41, 62, 67, 88, 152]. From an evolutionary point of view, this process is equivalent to acquiring traits that can lower the chances of elimination by natural selection. In natural ecosystems, species can be driven to extinction if they are less well-adapted to the existing environment than rival species [116]. This *survival-of-the-fittest* effect can be clearly observed in the app market. Specifically, similar to business firms competing in the market, apps are competing actors in an ecosystem of finite resources—only a handful of apps dominate downloads and revenue, leaving only a fraction of market value for other apps to compete over [25, 93, 133, 152].

In such an extremely competitive market, no app is too big to fail. In fact, recent app growth statistics have shown that the majority of apps lose around 80% of their users in the first 90 days of their intial release [101]. Therefore, app developers have to remain up-to-date with their users (consumers) needs in order to survive market selection. Staying close to the consumer not only minimizes the risk of failure, but also serves as a key factor in achieving market competence as well as managing and sustaining innovation. However, traditional approaches to solicit user feedback, such as surveys, opinion polls, and interviews, are often unable to cope with the scale of users in today'postive, nutrals app marketplace [94]. This has encouraged software providers to look beyond traditional software engineering practices into methods that enable them to connect with their end-users in a more effective and instant way. For instance, recent requirements engineering research

has focused on approaches that are designed to interact with the *crowd*, or the vast group of anonymous end-users who publicly share their experiences [27, 47, 55, 72, 106, 112]. The opinions of the crowd can be found wherever potential users discuss software, including app reviews, Twitter, online discussion forums, and other social media platforms. Crowd feedback contains important information that can help developers to understand user requirements and identify missing features [26, 27, 129]. Recent work in this domain has been focused on mining user reviews available on mobile app stores for technical feedback [27, 106]. Analysis of large datasets of app store user reviews has revealed that almost one third of these reviews contain useful information that can be translated into actionable software maintenance requests, such as feature requests (*"please add a dislike button"*) and bug reports (*"the app crashes every time I try to save"*) [27, 72, 106, 112]. However, despite these advances, existing work on soliciting crowd feedback for software requirements suffers from a number of limitations. These limitations can be described as follows:

- The majority of existing work has been focused on mining app store reviews for technical feedback [72, 106, 112, 131]. However, the number of reviews that are often available for individual apps is quite limited [115]. This emphasizes the need to look for other existing channels of feedback that app users often use to communicate with app developers. A popular channel for such information is the social media platform Twitter. From a software engineering perspective, Twitter has created an unprecedented opportunity for software providers to monitor the opinions of large populations of end-users of their systems [56]. Using Twitter, the crowd can publicly express their needs and concerns in the form of micro-blogs. Such data can be leveraged to extract rich and timely information about newly-released systems, enabling developers to get instant technical and social feedback about the performance of their apps.

- Mobile app user feedback classification techniques utilize general-purpose sentiment analysis techniques to annotate user posts (i.e., reviews and tweets) based on their

positive and negative sentiment. The main assumption is that the type (positive, neutral, and negative) and intensity of user sentiment can help text classifiers to separate urgent user issues more accurately [106]. However, existing state-of-the-art sentiment analysis techniques often fail to capture the emotional polarity of app users' feedback [75, 96]. The poor performance of these techniques (Stanford CoreNLP [154], SentiStrength [143], and NLTK [102]) can be attributed to the fact that they rely on the presence of generic English opinion lexicons and emotion-evoking words (e.g., *love, hate, like*) to detect emotions in text. However, software user feedback often includes computer jargon and neologisms (e.g., *brick, crash, fix, troll*). These words are typically overlooked by general-purpose sentiment analysis tools, thus, limiting their ability to function properly when applied to app user feedback [75, 96].

- Existing work on app user feedback analysis has been focused on mining user feedback at an individual app level. However, less attention has been paid to how such information can be extracted and synthesized at a domain level. An app domain can be defined as any collection of functionally-related apps. Specifically, the clusters of functionally-related apps form distinct *micro-ecosystems* within the app store ecosystem. A software ecosystem can be defined as a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them [69]. Systematically analyzing and synthesizing knowledge at the domain level is critical for developers to understand the current landscape of competition as well as get a more accurate picture of their end-users' expectations, preferences, and needs.

- Existing methods of review analysis are configured to classify, rather than model, user feedback. Models provide a framework for explicitly describing abstract salient concepts in a specific domain and formally reasoning about these concepts in order to create new knowledge [44, 65, 104]. Generating models for app domains can be crucial

for app developers to understand the complex interplay between the user, system, and business components of their ecosystem. Furthermore, existing research is mainly focused on isolating technical user concerns (e.g., bug reports and feature requests) in the reviews while ignoring the important business aspects of the ecosystem, thus leaving app developers totally unaware of other important non-technical issues in their domains. Extracting and modeling such concerns is very important to understand the intertwined dynamics of the ecosystem, especially in service oriented domains (e.g., food delivery) where multiple parties (e.g., restaurants, drivers, and consumers) are connected through the app.

To address these limitations, in this dissertation, we make the following contributions:

- In **Chapter 2**, we conduct a qualitative analysis to examine the value of Twitter as a source of technical user feedback that can be translated into actionable app maintenance requests. This analysis is conducted using 4,000 tweets collected from the Twitter feeds of 10 software systems, sampled from a broad range of application domains. In the second phase of our analysis, we employ text classification techniques to capture and categorize the various types of actionable software maintenance requests present in mobile apps' Twitter feeds. Furthermore, we investigate the performance of various text summarization techniques in generating compact summaries of the common concerns raised in technically informative tweets. The main goal is to support an adaptive data-driven software engineering process that can detect mobile app users' needs in an effective and timely manner.

- In **Chapter 3**, we present a preliminary analysis aimed at detecting and interpreting emotions found in software users' tweets. Specifically, through a case study of the new release of the iOS operating system, we demonstrate how the public opinion, as measured by Twitter sentiment, fluctuates in correspondence to specific software-related events. We further collect and manually classify 1,000 tweets for both sentiment

polarity, and software-specific expressions of emotion. We then evaluate the performance of multiple classification techniques in detecting emotions and collective mood states in software-relevant tweets and we compare their performance to other existing dictionary-based sentiment classifiers.

- In **Chapter 4**, we present a case study on the rise and fall of Yik Yak, one of the most popular social networking apps at its peak. In particular, we identify and analyze the design decisions that led to the downfall of Yik Yak, track other competing apps' attempts to take advantage of this failure, and perform user-driven feature-oriented domain analysis to understand and model users' concerns within the domain of anonymous social networking apps. The objective of the proposed analysis is to provide systematic guidelines for tracking and modeling the shift in app users' requirements and expectations and predicting the impact of feature updates on app user acquisition and retention rates.

- In **Chapter 5**, we propose an effective technique for automatically modeling crowd feedback at more focused categories of functionally-related mobile apps, or micro-ecosystems, in the mobile app store. To realize this goal, we present a case study targeting the domain of food delivery (courier) apps. Specifically, our analysis in this chapter is two-fold. First, we use several classification algorithms to categorize important crowd concerns in the ecosystem, and second, we propose and evaluate an automated technique for modeling these concerns along with their latent inter-dependencies. Our generated model can help app developers to stay aware of the most pressing issues in their ecosystem, and thus, develop sustainable release engineering strategies that can respond to these issues in an effective and timely manner.

- In **Chapter 6**, we conclude the dissertation and discuss prospects of future work.

Chapter 2. Twitter as a Source of App User Feedback

Twitter enables large populations of app users to publicly share their experiences and concerns about their apps in the form of micro-blogs. Such data can be collected and classified to help app developers infer users' needs, detect bugs in their code, and plan for future releases of their systems. However, automatically capturing, classifying, and presenting technically useful tweets is not a trivial task. Challenges stem from the scale of the data available, its unique format, diverse nature, and high percentage of irrelevant information and spam. Motivated by these challenges, this chapter reports on a three-fold study that is aimed at leveraging Twitter as a main source of app user feedback. The main objective is to enable a responsive, interactive, and adaptive data-driven release engineering process. Our analysis is conducted using 4,000 tweets collected from the Twitter feeds of 10 software systems sampled from a broad range of application domains. The results show that around 50% of collected tweets contain useful technical information. The results also show that text classifiers such as Support Vector Machines (SVM) and Naive Bayes (NB) can be very effective in capturing and categorizing technically informative tweets. Additionally, the chapter describes and evaluates multiple summarization strategies for generating meaningful summaries of informative software-relevant tweets.

2.1. Introduction

In recent years, Twitter has become one of the most popular micro-blogging social media platforms, providing an outlet for millions of users around the world to share their daily activities through real-time status updates. As of the fourth quarter of 2015, Twitter has averaged around 305 million monthly active users. The sheer volume of real-time and highly-diverse data provided by Twitter has revolutionized research in many data science areas. Twitter data has been leveraged to predict the daily ups and downs of the stock market [16], predict the political affiliation of the masses [31], and uncover and explain temporal variations in social happiness [38]. From a software engineering perspective,

Twitter has created an unprecedented opportunity for software providers to monitor the opinions of large populations of end-users of their systems [56].

Using Twitter, end-users of software can publicly express their needs and concerns in the form of micro-blogs. In fact, it has become a social media tradition that, with the release of each new mobile app, operating system, or web service, people resort to Twitter to describe their experiences and problems and recommend software to their friends, causing these systems to be *trending* worldwide. Such data can be leveraged to extract rich and timely information about newly-released systems, enabling developers to get instant technical and social feedback about their software.

Motivated by these observations, this chapter reports on a three-fold study that is aimed at leveraging Twitter as a main source of useful software user feedback. In particular, we evaluate the performance of multiple data classification and summarization techniques in automatically detecting and summarizing technical user concerns raised in software-relevant tweets. Automation is necessary to deal with the massive scale of Twitter data available, its unique format, and diverse nature [31]. Generated feedback can be used as an input for a well-informed release-planning process by pointing out critical bugs and helping developers prioritize the most desired features that need to be addressed in forthcoming releases [160]. This presents an advantage over classical user feedback collection methods that rely on face-to-face, user reviews, bug tracking, or survey communication. Ultimately, our main objective is to support an adaptive data-driven requirements engineering process that can detect users' needs in an effective and timely manner. In particular, in this chapter, we document the following contributions:

- We collect and manually classify 4,000 tweets sampled from the Twitter feeds of 10 different software systems. These systems extends over a broad range of application domains. Our objective is to qualitatively assess the technical value of software-relevant tweets.

- We employ two text classification techniques to accurately capture and categorize the various types of actionable software maintenance requests present in software systems' Twitter feeds.

- We investigate the performance of various text summarization techniques in generating compact summaries of the common concerns raised in technically informative tweets.

The remainder of this chapter is organized as follows. Section 2.2 motivates our work and describes our research questions. Section 2.3 describes our data collection and qualitative analysis process. Sections 2.4 and 2.5 evaluate the performance of various text classification and summarization strategies in capturing and summarizing technically informative tweets. Section 2.6 discusses the threats to the study's validity. Section 2.7 reviews related work. Finally, Section 2.8 concludes the paper and discusses prospects for future work.

2.2. Motivation and Research Questions

In this chapter we exploit the online micro-blogging service Twitter as a more open, more widespread, and more instant source of technically-informative software information. Unlike user reviews in mobile application stores, Twitter feedback is not limited to mobile applications. Rather, it extends to any software system with a sizable user base. Prior research on leveraging micro-blogging services in software engineering has been focused on the developer side, or how communities of software engineers use such services to support their daily development activities (Sec. 2.7.). Analysis of sample software-relevant tweets has revealed that developers frequently make use of social media as a means to facilitate team coordination, learn about new technologies, and stay in touch with the interests and opinions of all stakeholders [18, 139, 142, 155].

In our analysis, we examine the value of Twitter as a source of user feedback that can be translated into actionable software engineering requests. The main objective is to help

software developers to instantly and directly connect with their users, and ultimately, survive in a highly-competitive and volatile market. Based on these assumptions, we formulate the following research questions:

- **RQ_1: How informative is Twitter data for software engineers?** Twitter is a public service. Millions of tweets are generated every day by millions of users all over the world about a vast spectrum of subjects. The assumption that all these tweets carry useful technical information is unrealistic. For instance, users might tweet about software to share their experience with others, ask people to follow them on social media platforms, or recommend a video game to their friends. Therefore, the first objective of our analysis is to determine how technically informative, or useful, software users' tweets are. Technically informative tweets can be described as any user concern that can be translated into an actionable software maintenance request, such as a bug report or a user requirement. Uninformative tweets, on the other hand, can be simply spam or messages with no immediate technical feedback to the developer.

- **RQ_2: To what extent can informative software-relevant tweets be automatically classified?** Assuming software-relevant tweets contain sufficient user technical feedback, can such information be automatically captured? Manually filtering through massive amounts of Twitter data can be a laborious and error-prone task. Therefore, for any solution to be practical, automated support is needed to facilitate a more effective data filtering process that can separate, with a decent level of accuracy, technically informative from uninformative tweets.

- **RQ_3: How can informative tweets be accurately summarized?** Twitter posts are lexically and semantically restricted. Furthermore, several tweets might raise similar concerns. Presenting such large and heavily redundant amounts of raw tweets to developers can cause confusion. This emphasizes the need for automated

methods to summarize informative tweets in such a way that enables a more effective data exploration process.

2.3. Data Collection and Qualitative Analysis

In this section, we answer our first research question regarding the potential value of tweets for software developers. In particular, we describe our data collection process along with the main findings of our manual qualitative analysis.

2.3.1. Data Collection

In our analysis, we used Twitter's Search API to collect our dataset [2]. This API can be customized to search for a specific word or hashtag (#) in Twitter feeds. Twitter has integrated hashtags into the core architecture of the service, allowing users to search for these terms explicitly to retrieve a list of recent tweets about a specific topic. Searching through hashtags can be effective when Twitter is mined at a massive scale to infer the opinions of the masses towards a certain topic, such as learning peoples' views of a certain public figure (search for *#obama*) or certain recent events (*#election*) [38, 128]. However, one of the main drawbacks of hashtag, or word, search is the very high noise-to-signal ratio. More specifically, such queries can return millions of tweets. For example, a search for *#snapchat* returns millions of tweets of the nature "please follow me on #snapchat". While such vast amounts of data can be very useful for inferring public trends, classifying such data manually can be a tedious task. To overcome these limitations, in our analysis, we limit our data collection process to tweets addressed directly to the Twitter account of a given software product (e.g., tweets beginning with *@Windows10*). This strategy ensures that only tweets meant to be a direct interaction with the software provider are included.

We selected 10 software products from a broad range of domains to conduct our analysis (Table 2.1). The data collection process was repeated on a daily basis from April 1st to May 27th of 2016. The resulting dataset contained 188,737 unique tweets. Figure 2.1 shows the daily number of tweets collected over the course of our data collection process. Random

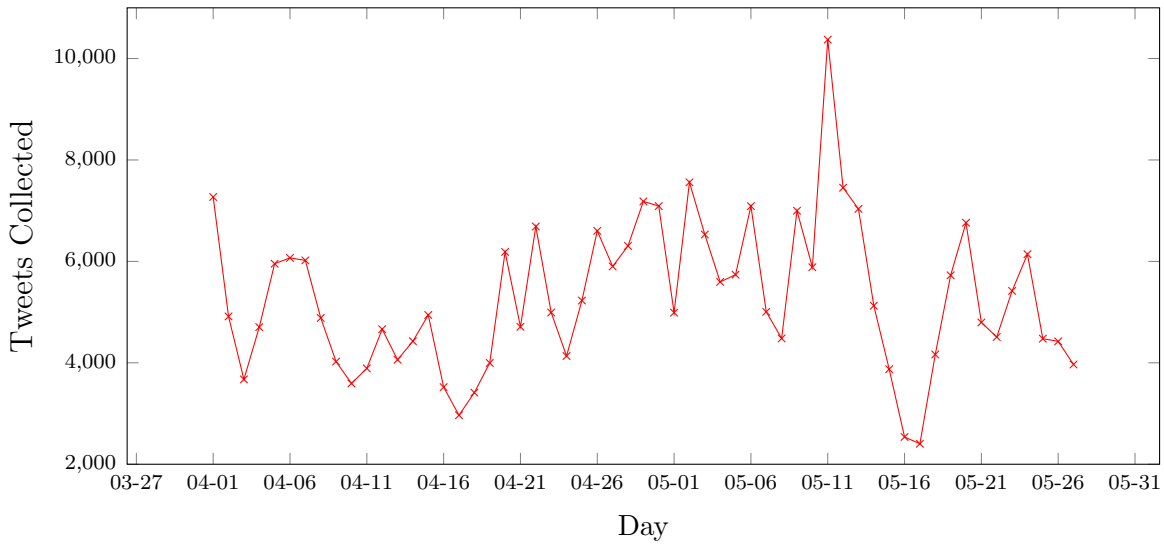

Figure 2.1. Number of tweets collected per day (April 1st - May 27th, 2016)

sampling was used to prepare our dataset. In particular, a Ruby script is used to randomly select 400 tweets from the set of tweets collected for each software system.

2.3.2. What's in a tweet?

To get a sense of the information value of our data (i.e., to answer $\mathbf{RQ_1}$), the sampled data is manually analyzed. To conduct our analysis, we adopt the standard categorization typically used to classify user reviews in app store mining research [106, 129]. More specifically, we assume that tweets addressed to the account of a software system can be classified into two main categories, including technically informative and uninformative messages.

Our manual classification process was performed by three independent industry professional experts with an average of 5 years of experience in software development. Each expert examined each sampled tweet in our dataset. A majority vote was taken in cases of a conflict. Conflicts arise in cases that carry a double meaning. For example, the tweet "@android run app and it dies, any way to know why it dies rather than looking through logs?" could be classified as a bug report (i.e., *it dies*) or a user requirement (i.e., *any way to*). In total, 121 conflicts were detected in our data ($\approx 3\%$). Table 2.1 summarizes our findings. The outcome of our manual classification process can be described as follows:

Table 2.1. Our sample systems and number of bug, requirement, and miscellaneous tweets collected for each system

System	domain	Bugs	Req.	Other
Android	Operating Systems	69	66	265
Apple Support	Customer service	179	47	174
Call of Duty	Gaming	44	121	235
Google Chrome	Browsing	150	61	189
Instagram	Social Media	87	98	215
Minecraft	Gaming	61	86	253
Snapchat	Social Media	80	172	148
Visual Studio	Development	136	86	178
WhatsApp	Messaging	112	118	170
Windows 10	Operating Systems	143	94	163
Total		1061	949	1990

- **Bug reports:** These tweets report a potential problem with the software. For example, "@googlechrome I have never ever seen the auto-update function of chrome work on any of all my computers." and "@Photoshop When will the biggest Photoshop issue of: lag, freezing, unresponsive Marquee Tool be fixed?".

- **User requirements:** These tweets mainly include requests for new features, or alternatively express that a recently added feature is undesirable. For example "@Snapchat pls make it to where I can see an individual score with someone so I know how many snaps we've sent back & forth !". Some requests tend to be less obvious, especially requests for a removed feature to be added back, for example "@googlechrome any chance I could get my bookmark folders back now please?". Such requests can play a crucial rule in release planning as they help developers to decide what features to include/omit in the new release.

- **Miscellaneous and spam:** A considerable part of software-relevant tweets do not provide any useful technical information to the developer. Such tweets might include praise (e.g. "@VisualStudio is quickly becoming my #goto #dev environment"),

insults (e.g. "mediocre as usual #meh"), general information or news (e.g. "@WhatsApp announced it's started full end-to-end #encryption across its messaging app."), and spam. Spammers take advantage of the openness and popularity of Twitter to spread unsolicited messages to legitimate users [114]. For instance, spammers tend to post tweets containing typical words of trending topics along with URLs that lead users to completely unrelated websites (e.g. "#imgur #fix #problem bit.ly/1xTYs").

In summary, our manual analysis shows that, out of the 4000 tweets examined, 51% of these tweets were technically informative (27% bug reports and 24% user requirements), while the other 49% were basically spam and miscellaneous information. These findings answer $\mathbf{RQ_1}$ and motivate $\mathbf{RQ_2}$ and $\mathbf{RQ_3}$. In particular, given that around half of our data contain potentially useful information, how can that information be automatically identified and effectively summarized?

2.4. Automatic Classification

The second phase of our analysis is concerned with automatically classifying our ground-truth dataset into the different categories of tweets identified earlier. Our research question under this phase ($\mathbf{RQ_2}$) can be broken down into two sub-questions, first, what classifiers are most effective in the context of Twitter data, and second, what classification features generate the most accurate results.

2.4.1. Classifiers

To answer the first part of $\boldsymbol{RQ_2}$, we investigate the performance of two text classification algorithms, including Naive Bayes (NB) and Support Vector Machines (SVM). SVM and NB have been found to work well with short text. Short-text is a relatively recent Natural Language Processing (NLP) type of text that has been motivated by the explosive growth of micro-blogs on social media (e.g., Tweets and YouTube and Facebook comments) and the urgent need for effective methods to analyze such large amounts of lexically and semantically limited textual data [31, 114, 161]. For instance, Twitter posts are limited to 140 character long messages (*tweets*) and typically contain colloquial terms (e.g., *LOL*,

smh, *idk*), hyperlinks, Twitter-specific characters such as hashtags (#) and mentions (@), along with phonetic spellings and other neologisms [146]. In detail, NB and SVM can be described as follows:

- **Naive Bayes (NB):** NB is an efficient linear probabilistic classifier that is based on Bayes' theorem [85]. NB assumes the conditional independence of the attributes of the data. In other words, classification features are independent of each other given the class. In the context of text classification, the features of the model can be defined as the individual words of the text. Under this approach, known as the *Bag-of-Words* (BOW), the data is typically represented by a 2-dimensional *word x document* matrix. In the Bernoulli NB model, an entry in the matrix is a binary value that indicates whether the document contains a word or not (i.e., {0,1}). The Multinomial NB, on the other hand, uses normalized frequencies of the words in the text to construct the *word x document* matrix [113].

- **Support Vector Machines (SVM):** SVM is a supervised machine learning algorithm that is used for classification and regression analysis in multidimensional data spaces [24]. SVM attempts to find optimal hyperplanes for linearly separable patterns in the data and then maximizes the margins around these hyperplanes. Technically, support vectors are the critical instances of the training set that would change the position of the dividing hyperplane if removed. SVM classifies the data by mapping input vectors into an N-dimensional space, and deciding on which side of the defined hyperplane the data instance lies. SVMs have been shown to be effective in domains where the data is sparse and highly dimensional [74].

2.4.2. Classification Features

To augment Twitter's uniquely limited form of text communication, researchers typically use combinations of additional features to help the classifier make more accurate decisions [74, 147, 163]. These features include:

- **Textual Content (BOW)**: Our main classification feature is the words of the tweet. The phrase *Bag-of-Words* (**BOW**), stems from the fact that the text is simply represented as an un-ordered collection of words. Given that Twitter limits messages to 140 characters, a tweet typically has 14 words on average.

- **Text processing**: This set of features includes text reduction strategies such as stemming (**ST**) and stop-word (**SW**) removal. Stemming reduces words to their morphological roots. This leads to a reduction in the number of features (words) as only one base form of the word is considered. Stop-word removal, on the other hand, is concerned with removing English words that are considered too generic (e.g., *the*, *in*, *will*). We further remove words that appear in one data instance (tweet) since they are highly unlikely to carry any generalizable information [31, 53].

- **Sentiment Analysis (SA)**: Sentiment analysis is concerned with determining whether a text conveys positive or negative feelings. Sentiment analysis has been found to play a paramount role in Twitter data analytics [130]. For instance, specific Twitter moods, or sentiments, were found to correlate with public opinion regarding subjects such as consumer confidence and political affiliation, even predicting the movement of the stock market [16, 128]. In our analysis, we assume that a negative sentiment might be associated with a bad experience, such as a system failure or a bad feature. A positive sentiment, on the other hand, might indicate a positive experience [129].

2.4.3. Evaluation

To implement NB and SVM, we use Weka [3], a data mining suite that implements a wide variety of machine learning and classification techniques. We also use Weka's built-in

stemmer (`IteratedLovinsStemmer` [103]) and stop-word list to preprocess the tweets in our dataset. In our analysis, we use Multinomial NB, which uses the normalized frequency (TF) of words in their documents [113]. Multinomial Naive Bayes is known to be a robust text classifier, consistently outperforming the binary feature model (Multi-variate Bernoulli) in highly diverse, real-world corpora [113]. SVM is invoked through Weka's SMO, which implements John Platt's sequential minimal optimization algorithm for training a support vector classifier [134]. In our analysis, the best results were obtained using the Pearson VII function-based universal kernel (*Puk*) with kernel parameters $\sigma = 8$ and $\omega = 1$ [158]. Universal Kernels are known to be effective for a large class of classification problems, especially for noisy data [148].

Sentiment analysis is performed using Sentistrength [1]. Sentistrength analyzes a document and assigns it two values: a positive sentiment strength and a negative sentiment strength [126, 159]. In particular, the input text is rated by the sentiment content of each word. Most words are neutral, but some words, such as *loved* or *hated* increase the respective positive or negative sentiment score for their sentence. In our analysis, a text is basically a tweet. For example, the tweet "@googlechrome really is the best option for someone who enjoys platform agnosticism. All my stuff in @Windows and #OSX" receives a positive score of 3 and a negative score of 1. The positive score originates from the words *best* (1 point) and *enjoys* (2 points) and the negative score of 1 point is the default score for texts with no negative terminology.

To train our classifiers, we use 10-fold cross validation. This method creates 10 partitions of the dataset such that each partition has 90% of the instances as a training set and 10% as an evaluation set. The benefit of this technique is that it uses all the data for building the model, and the results often exhibit significantly less variance than those of simpler techniques such as the holdout method (e.g., 70% training set, 30% testing set).

Recall, precision, and F-measure are used to evaluate the performance of the different classification techniques used in our analysis. Recall is a measure of coverage. It represents

Table 2.2. Summary of classification accuracy

	Bug Rep.			User Req.		
Configurations	**P**	**R**	**F**	**P**	**R**	**F**
NB:	0.74	0.77	0.76	0.77	0.58	0.66
NB + ST	0.71	0.80	0.75	0.74	0.58	0.65
NB + SW + ST	0.70	0.75	0.72	0.70	0.54	0.61
NB + Sent.	0.75	0.76	0.75	0.78	0.56	0.65
SVM	0.79	0.75	0.77	0.72	0.60	0.65
SVM + ST	0.78	0.77	0.78	0.72	0.61	0.66
SVM + SW + ST	0.77	0.69	0.72	0.70	0.58	0.63
SVM + Sent.	0.78	0.76	0.77	0.72	0.60	0.66

the ratio of correctly classified instances under a specific label to the number of instances in the data space that actually belong to that label. Precision, on the other hand, is a measure of accuracy. It represents the ratio of correctly classified instances under a specific label to the total number of classified instances under that label. Formally, if A is the set of data instances in the data space that belong to the label λ, and B is the set of data instances that were assigned by the classifier to that label, then recall ($\boldsymbol{R}$) can be calculated as $R_\lambda = |A \cap B|/|A|$ and precision ($\boldsymbol{P}$) can be calculated as $P_\lambda = |A \cap B|/|B|$. We also use $F = 2PR/(P + R)$ to measure the harmonic mean of recall and precision.

2.4.4. Results and Discussion

The results of our classification process are shown in Table 2.2. In terms of classifiers' accuracy, on average, both SVM and NB were able to achieve competitive results. These results can be explained based on the characteristics of our data space. More specifically, even though Twitter messages are limited in size, the feature space (number of words) is typically very large [74]. This can be attributed to the fact that people use informal language (e.g., slang, acronyms, and abbreviations) in their tweets. This drastically increases the number of features the classifier needs to process, and also leads the vector representation (BOW) of Twitter messages to be very sparse. While machine learning algorithms

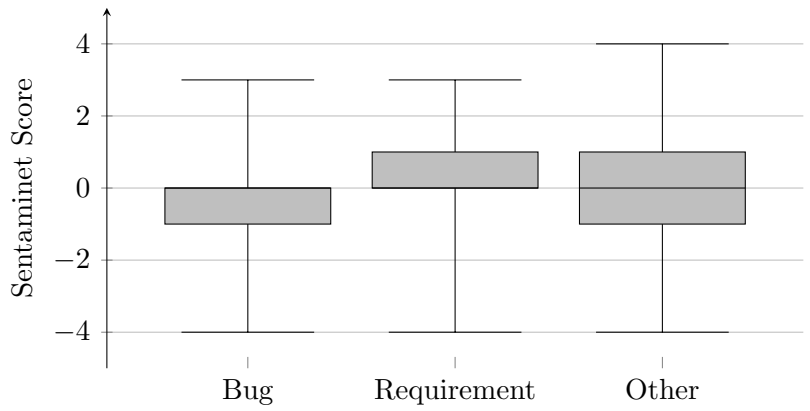

Figure 2.2. Sentiment scores for bug reports, user requirements, and other tweets

tend to over-learn when the dimensionality is high, SVM has an over-fitting avoidance tendency—an inherent behavior of margin maximization which does not depend on the number of features [23]. Therefore, it has the potential to scale up to high-dimensional data spaces with sparse instances. NB tends to be more robust to noise, which seems to work in Twitter data classification, despite the conditional independence assumption among classification features [163].

In terms of classification features, the results also show that sentiment scores had almost no impact on performance. This can be attributed to the fact that, unlike political tweets which tend to be very polarized, and typically carry intense emotions [31], software-relevant tweets tend to be neutral. To gain more insight into these results, the boxplots in Figure 2.2 show the average combined sentiment score of the different classes of tweets averaged over all our software systems. The figure shows that while bugs tend to be slightly negative (-1), and requirements slightly positive (+1), the difference was not enough to affect the classification accuracy. Furthermore, miscellaneous tweets, while they might carry some extreme emotions, also tend to average out to a neutral state (-1, +1).

Our results show that text processing features, such as stemming (**ST**) and stop-word removal (**SW**), produced mixed results for different classifiers. On the one hand, SVM's performance was slightly enhanced when stemming was applied, while NB performance slightly dropped. On the other hand, removing English stop-words seems to have a more

noticeable negative impact on the results. In general, stop words seem to add important information value to the classifier. For instance, some of these words (e.g., *would*, *should*, *will*, *don't*, *please*) actually represent distinctive features of user requirements and bug reports (e.g., "@Snapchat would you please bring the dog filter back?"). Therefore, removing such words leads to a decline in the classification accuracy.

In summary, to answer $\mathbf{RQ_2}$, our results show that in the context of software relevant tweets, the textual content of Twitter messages is the only contributing factor to the classification accuracy. Other features that are often used as supplemental attributes to enhance Twitter data are irrelevant. The results also confirm previous findings regarding the suitability of SVM and NB as robust classifiers for Twitter data [163].

2.5. Summarization

The third phase of our analysis is focused on generating succinct summaries of the technically useful software tweets. A summary can be described as a compact description that encompasses the main theme of a collection of tweets related to a similar topic [80, 100].

2.5.1. Tweets Summarization: A Pilot Study

The summarization task in our analysis can be described as a multi-document summarization problem, where each tweet is considered as a separate document. In general, multi-document summarization techniques can be either extractive or abstractive. Extractive methods select specific documents, or keywords, already present within the data as representatives of the entire collection. Abstractive methods, on the other hand, attempt to generate a summary with a proper English narrative from the documents. Generating abstractive summaries can be a very challenging task. It involves heavy reasoning and lexical parsing to paraphrase novel sentences around information extracted from the corpus [59]. This problem becomes more challenging when dealing with the lexically and semantically limited Twitter messages. Therefore, extractive summarization techniques are typically employed to summarize micro-blogging data [125].

To get a sense of how developers would identify the main topics in a list of software-relevant tweets, we conducted a pilot study using two expert programmers with more than 10 years of programming experience each. Each expert was assigned 4 sets of tweets, including 2 sets of bug reporting tweets and 2 sets of user requirement tweets from 4 different systems, including: *Snapchat*, *Chrome*, *Whatsapp*, and *Windows10*. Their task was to go through each set and identify ten tweets that they thought captured the main topics raised in the set. No time constraint was enforced.

Our experts were interviewed after the experiment. Both of them implied that they initially identified the main topics in the tweets after going through the set once or twice. Once these topics were identified (based on their frequent appearance), they selected tweets that included requests (terms and phrases) related to the main topics identified. Our experts were then provided with a word cloud for each set of tweets they were asked to summarize. Each cloud includes the most frequent 30 terms from each set, where more frequent words are drawn in larger font. Our experts were then asked if they thought these clouds were sufficient to convey the main concerns raised in the set. Both experts implied that they preferred full tweet summaries over keyword summaries. In general, word-clouds lack context and structure. In contrast, full tweets have the advantage of being full sentences, and thus, carry more meaningful information [12, 80]. For example, examining the set of feature requests addressed to *Snapchat* shows that they revolve around three main concerns, including users asking for new *filters* to be added, users complaining about the *auto-play* feature of Snapchat stories, and a few tweets raising *portability* concerns (requesting *Snapchat* to work on other platforms). Table 2.3 shows examples of the tweets related to the `automatic play` feature of Snapchat stories and the `filter` feature. Extracting full tweets gives developers a better idea of what the common user concerns actually are. However, only displaying tags (individual words) might be not as informative.

Based on our observations during our pilot study, in our analysis we examine the performance of frequency-based extractive summarization techniques in summarizing software

Table 2.3. Example of requirement tweets related to similar features from Snapchat's Twitter feed

Tweets related to the feature ``filter''	**Tweets related to the feature ``automatic play''**
"where is my dog filter"	"change the way you view stories back to the original way?"
"can you bring back the bunny face filter pls"	"I do not want to automatically watch people's stories!!"
"more arty filters like the one today"	"hate the stories autoplay feature"

relevant tweets. These techniques rely on the frequencies of words as an indication of their perceived importance [121]. In other words, the likelihood of words appearing in a human-generated summary is positively correlated with their frequency [68]. Formally, a full-tweet extractive summarization process can be described as follows: given a topic keyword or phrase M and the desired length for the summary K, generate a set of representative tweets T with a cardinality of K such that $\forall t_i \in T, M \in t_i$ and $\forall t_i, \forall t_j \in T, t_i \nsim t_j$. The condition $t_i \nsim t_j$ is enforced to ensure that selected tweets provide sufficiently different information (i.e., are not redundant) [68].

In our analysis, we investigate the performance of a number of extractive summarization techniques that have been shown to work well in the context of mirco-blogging data on social media [68, 80, 118, 121]. These techniques include:

- **Hybrid Term Frequency (TF)**: Hybrid TF is the most basic method for determining the importance of a tweet. Formally, a word's w_i value to the summary is computed as the frequency of the word in the entire collection of tweets $f(w_i)$ divided by the number of unique words in the collection (N). This *hybrid* modification over classical single-document TF is necessary to capture concerns that are frequent over the entire collection [68]. The probability of a tweet of length n words to appear in the summary is the average of the weights of its individual words: $\frac{1}{n}\sum_{i=1}^{n} f(w_i)/N$.

- **Hybrid TF.IDF**: Introduced by Inouye and Kalita [68], the hybrid TF.IDF approach is a frequency-based summarization technique that is designed to summarize social media data. Hybrid TF.IDF accounts for a word's scarcity across all the tweets by using the inverse document frequency (IDF) of the word. IDF penalizes words that

are too frequent in the text. Formally, TF.IDF can be computed as:

$$TF.IDF = TF(w_i) \times \log \frac{|D|}{|d_j : w_i \in d_j \wedge d_j \in D|} \tag{2.1}$$

where $TF(w_i)$ is the term frequency of the word w_i in the entire collection, $|D|$ is the total number of tweets in the collection, and $|d_j : w_i \in d_j \wedge d_j \in D|$ is the number of tweets in D that contain the word w_i. The importance of a tweet can then be calculated as the average TF.IDF score of its individual words.

To control for redundancy, or the chances of two very similar tweets getting selected, before adding a top tweet to the summary, the algorithm makes sure that the tweet does not have a textual similarity above a certain threshold with the tweets already in the summary. Similarity is calculated using the cosine between the vector representations of tweets.

- **SumBasic**: Introduced by Nenkova and Vanderwende [121], SumBasic uses the average term frequency (TF) of tweets' words to determine their value. However, the weight of individual words is updated after the selection of a tweet to minimize redundancy. This approach can be described as follows:

  1. The probability of a word w_i in the input corpus of size N words is calculated as $\rho(w_i) = f(w_i)/N$, where $f(w_i)$ is the frequency of the word in the entire corpus.
  2. The weight of a tweet of length n words is calculated as the average probability of its words, given by: $\frac{1}{n} \sum_{i=1}^{|n|} \rho(w_i)$
  3. The best scoring tweet is selected. For each word in the selected tweet, its probability is reduced by $\rho(w_i)_{new} = \rho(w_i) \times \rho(w_i)$. This step is necessary to control for redundancy, or minimize the chances of selecting tweets describing the same topic with high frequency words.
  4. Repeat from 2 until the required length of the summary is met.

2.5.2. Evaluation

We recruited 10 programmers (experts) to participate in our experiment, including 3 graduate students in computer science and 7 industry professionals. Our experts have reported an average of 6 years of programming experience. 5 systems from Table 2.1 were randomly selected to conduct our experiment. These systems include: *Chrome*, *Minecraft*, *SnapChat*, *Whatsapp*, and *Windows 10*. Each of our experts was assigned 2 different systems to summarize, such that, each system is summarized by exactly 4 different experts. For each system we provided two sets of tweets, including the set of bug reporting tweets and the set of user requirement tweets. The main task of the expert was to go through each set and identify 10 tweets that they believed captured the common concerns raised in the set. The tweets in each set were randomized ahead of time to avoid any ranking bias (e.g., an expert would always favor tweets from the top of the list). No time constraint was enforced. However, most of our participants responded within a one week period.

The various summarization techniques proposed earlier were then used to generate the automated summaries for the 5 systems included in our experiment. To enhance the quality of the generated summaries, English stop-words were excluded from our frequency analysis. Stemming was also applied to minimize the redundancy imposed by the usage of different variations of words (e.g., *show, showing, shown, and shows*). To assess the quality of these summaries, for each system, we calculate the average term overlap between our experts' selected lists of tweets (reference summaries) and the various automatically generated summaries. Formally, a recall of a summarization technique t is calculated as:

$$Recall_t = \frac{1}{|S|} \sum_{i=1}^{|S|} \frac{match(t, s_i)}{count(s_i)} \tag{2.2}$$

where S is the number of reference summaries, $match(t, s_i)$ is the number of terms that appear in the reference summary s_i and the automated summary generated by t, and $count(s_i)$ is the number of unique terms in the reference summary s_i. An automated

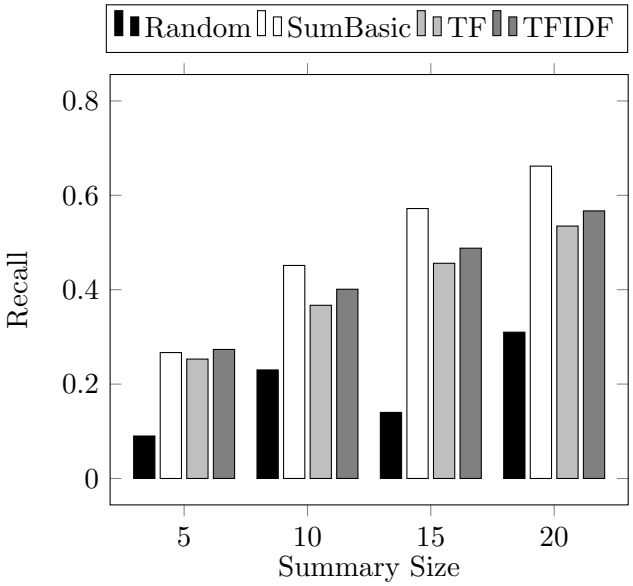

Figure 2.3. Performance of different summarization techniques over bug reporting tweets measured at different length summaries (5, 10, 15, 20)

summary that contains (recalled) a greater number of terms from the reference summary is considered more effective [68, 97, 121]. In our analysis, recall is measured over different length summaries (5, 10, 15, and 20 tweets included in the summary).

2.5.3. Results and Discussion

The recall of the different summarization techniques is shown in Figure 2.3 and Figure 2.4. Randomly generated summaries (tweets were selected randomly from each set using the `.NET Random` class) were used to compare the performance of our proposed methods. Our results show that all methods outperformed the random baseline. On average, SumBasic was more successful than hybrid TF.IDF and TF in summarizing the common concerns found in software-relevant tweets. TF achieved the poorest performance, suggesting that redundancy control is important in order to achieve comprehensive summaries. The results also show that TF was only slightly outperformed by hybrid TF.IDF. Due to the limited nature of the documents in our corpus (i.e., individual tweets), the IDF part seems to have a limited impact on the results as it is typically dominated by TF.

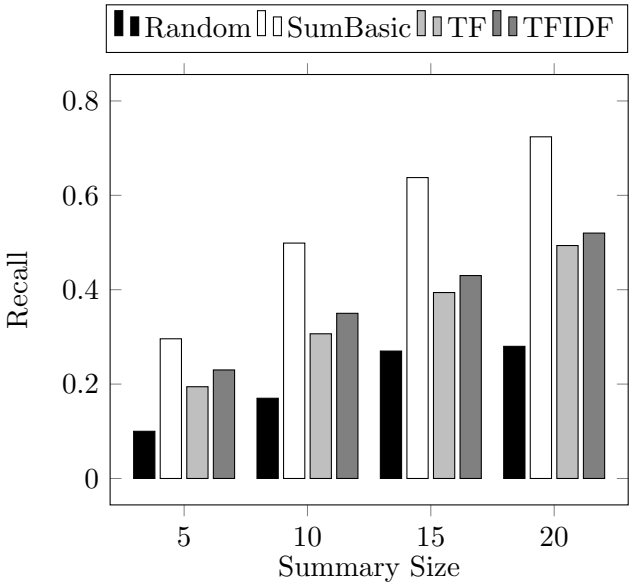

Figure 2.4. Performance of different summarization techniques over user requirements tweets measured at different length summaries (5, 10, 15, 20)

The better performance of SumBasic in comparison to hybrid TF.IDF can be explained based on their redundancy control mechanisms. SumBasic tends to be more forgiving for redundant terms as it avoids excessive redundancy while also allowing common words to occasionally repeat in the summary. This can be useful in cases where there is relatively high redundancy in the data. Hybrid TF.IDF, by enforcing a similarity threshold on the entire tweet rather than individual words, can exclude important concerns from the summary. More specifically, changing the redundancy threshold can lead to extreme changes in the summaries (excluding many tweets or hardly any). Finding an optimal threshold that works for all cases can be an exhaustive task especially that different datasets might require different thresholds. For instance, the best hybrid TF.IDF recall in our analysis was observed at similarity thresholds of 0.65 for bug reports and 0.50 for user requirements.

It is important to point out that more computationally expensive techniques such as Latent Dirichlet Allocation (LDA) [15] and cluster-based summarization have been employed in the literature to summarize Twitter data [80]. However, due to the lack of semantic structure in Twitter posts, such complex relational models were reported to be

ineffective in capturing topical information. Furthermore, such techniques typically require heavy calibration of multiple parameters in order to generate decent results. This limits the practicality of such techniques and their ability to produce meaningful summaries [12, 100]. Frequency-based techniques, on the other hand, are computationally inexpensive and relatively easier to implement and calibrate. This aspect can be crucial to achieve an easy transfer of our research findings to practice.

2.6. Threats to Validity

The study presented in this chapter has several limitations that might affect the validity of the results [36]. A potential threat to the proposed study's internal validity is the fact that human judgment is used to classify and summarize our sample tweets and prepare our ground-truth dataset. This might result in an experimental bias as humans tend to be subjective in their judgment. However, it is not uncommon in text classification to use humans to manually classify the data, especially in social media classification [18, 155]. Similarly, evaluating machine-generated against human-generated summaries is a standard evaluation procedure. Therefore, while the subjectivity and bias threats that stem from using humans are inevitable, they can be partially mitigated by using multiple judges at different levels of expertise.

In our experiment, there were minimal threats to construct validity as the standard performance measures (Recall, Precision), which are extensively used in related research, were used to assess the performance of different methods.

Threats to external validity impact the generalizability of results [36]. A potential threat to our external validity stems from the fact that our dataset is limited in size and was generated from a limited number of software systems and tweets. To mitigate this threat, we ensured that our dataset was compiled from a wide variety of application domains. Furthermore, we used randomization to sample 400 tweets from the set of tweets collected for each system. While considering all the data instances in the analysis might enhance the validity of our results, manually analyzing such large amounts of data can be

a tedious and error-prone task, and as a result, research on Twitter data analytics is often conducted using partial datasets. Other threats might stem from the tools we used in our analysis. For instance, we used Weka as our machine learning and classification platform and Sentistrength was used to tag our tweets' sentiment. Nonetheless, such tools have been extensively used in the literature and have been shown to generate robust results across a plethora of applications. Furthermore, using such publicly available benchmark tools enables other researchers to replicate our results.

Finally, it is unclear whether our approach will be as successful for systems that are less widely used. More specifically, there is no guarantee that such systems will have enough tweets that offer meaningful data for developers. In such cases, other sources of user feedback, such as app store reviews and online surveys, can be used to paint a full picture. Another concern stems from the fact that Twitter is often used for advertisement purposes by software vendors. Many tweets can be simply marketing messages from the vendor or competitors, rather than feedback from users. Therefore, there is no guarantee that the technically informative tweets captured by our approach are indeed from the right audience and not just containing publicity which accidentally describes as a requirement or a bug report.

2.7. Related Work

In software engineering, the research on mining micro-blogging services has focused on the way developers use such platforms to share and exchange software development information. For instance, Bougie et al. [18] manually analyzed 600 tweets from three different software engineering communities. The results showed that developers' tweets tend to include more conversation and information sharing in comparison to non-technical populations of Twitter users.

Tian et al. [155] manually analyzed a sample of 300 developer tweets in various software engineering communities. The results showed that such tweets commonly contain job openings, news, questions and answers, or links to download new tools and code. In a

follow-up study, Prasetyo et al. [136] investigated the feasibility of automatically classifying tweets as relevant and irrelevant to software developers in engineering software systems. The authors used SVM to classify the data in [155]. The results showed that 47% of the classified tweets were found to be relevant to developers.

Singer et al. [142] surveyed 271 and interviewed 27 active GitHub developers about using Twitter in their development and interaction activities. The authors reported that developers use Twitter mainly to stay aware of industry changes, for learning, and for building relationships. Sharma et al. [139] proposed an approach to help developers to identify software relevant tweets. Individual tweets were assigned a relevance probability based on their similarity to a language model generated from a subset of posts from *Stack-Overflow*. Evaluating the proposed approach over a random sample of 200 tweets showed improvement over previous models that use classification and keyword dictionaries.

Initial exploratory work on the value of Twitter data for requirements engineers was proposed by Guzman et al. [56]. The authors manually analyzed and classified a sample of 1,000 tweets to determine the usage characteristics and content of software-relevant tweets. The results showed that software tweets contained useful information for different groups of technical and non-technical stakeholders. While our work builds upon this work, in our analysis we only focused on feedback dedicated to the technical stakeholders of the system (developers) by limiting data collection to tweets directly addressed to the Twitter accounts of our sample systems. This constraint enabled us to obtain higher accuracy levels. For instance, qualitative analysis of our 4,000 tweets showed that around 50% of our data contained useful technical feedback, in comparison to only 19% in [56]. We were also able to achieve an average classification F_1 of 72% using SVM, in comparison to an F_1 of 48% achieved in [56].

2.8. Conclusions and Future Work

This chapter presents a three-fold procedure aimed at leveraging Twitter as a main source of technical software information. These phases include data collection, classifica-

tion, and presentation. Our analysis is conducted using 4,000 tweets sampled from tweets addressed to 10 software systems from a broad range of application domains. A manual qualitative analysis was conducted to determine the information value of sampled tweets. Our results showed that around 50% of these tweets contained useful technical data that can be translated into actionable bug reports and user requirements ($\mathbf{RQ_1}$). Our analysis also showed that SVM and NB can be effective in capturing and categorizing technically useful tweets. In terms of classification features, our results showed that in the context of software-relevant tweets, sentiment analysis seems to have no impact on performance, while text reduction strategies had a conflicting impact on the classification accuracy ($\mathbf{RQ_2}$).

Technically informative tweets were then summarized using multiple automated summarization algorithms. These algorithms, including hybrid TF, hybrid TF.IDF and SumBasic, are known for their simplicity (implementation, calibration, and computation overhead) and decent performance in the context of social media data. A human experiment using 10 programmers was conducted to assess the performance of the different summarization techniques. The results showed that the summarization algorithm SumBasic was the most successful in recalling majority of the common concerns raised in software-relevant tweets ($\mathbf{RQ_3}$).

The line of work in this chapter will be expanded along several directions as follows:

- Data collection: A main part of our future effort will be devoted to collecting larger datasets from a more diverse set of software systems. More data will enable us to conduct in depth analysis of software users' tweeting patterns, and thus draw more robust conclusions.

- Analysis: Our future work will include experimenting with more advanced text classification and summarization techniques to achieve higher levels of accuracy.

- Tool support: A working prototype that implements our findings in this chapter will be developed. This prototype will enable developers to classify and summarize tweets related to their systems in an effective and accurate manner.

Chapter 3. Sentiment Analysis of App User Feedback

Twitter enables software developers to track users' reactions to newly released systems. Such information, often expressed in the form of raw emotions, can be leveraged to enable a more informed software release process. However, automatically capturing and interpreting multi-dimensional structures of human emotions expressed in Twitter messages is not a trivial task. Challenges stem from the scale of the data available, its inherently sparse nature, and the high percentage of domain-specific words. Motivated by these observations, in this chapter we present a preliminary study aimed at detecting, classifying, and interpreting emotions in software users' tweets. A dataset of 1000 tweets sampled from a broad range of software systems' Twitter feeds is used to conduct our analysis. Our results show that supervised text classifiers (Naive Bayes and Support vector Machines) are more accurate than general-purpose sentiment analysis techniques in detecting general and specific emotions expressed in software-relevant Tweets.

3.1. Introduction

Emotions in Twitter messages can be detected using sentiment analysis techniques. Sentiment analysis is concerned with determining whether a text conveys positive or negative feelings. In general, sentiment analysis techniques rely on the presence of English opinion lexicons and emotion-evoking words (e.g., *love, hate, like*) to detect feelings in text. However, software relevant tweets often include computer jargon words (e.g., *brick, uninstall, fix, and crash*). These words carry multi-dimensional structures of positive and negative human emotions that are typically overlooked by general-purpose sentiment analysis methods. To address these challenges, in this paper we present a preliminary analysis aimed at detecting and interpreting emotions present in software-relevant tweets. Our analysis is conducted using a dataset of 1000 tweets sampled from the Twitter feeds of a broad range of software systems. Our objectives are to **a)** identify the most effective techniques in detecting emotions and collective mood states in software-relevant tweets, and **b)** investigate how such emotions are correlated with specific software-related events.

The remainder of this chapter is organized as follows. Section 3.2 describes our research problem and motivates our work. Section 3.3 presents our preliminary experimental analysis and discusses our main findings and their potential impact. Finally, Section 3.4 concludes the chapter and describes our prospects of future work.

3.2. Background and Motivation

Researchers in behavioral economics, politics, and social studies have reported that emotions play a significant role in human decision-making [35]. Such information is typically collected through face-to-face, email, poll, or survey communication. In recent years, Twitter has emerged as a more instant and a more wide-spread source of public information that can complement traditional data collection methods. Tweets analyzed through sentiment analysis techniques can be used to infer the public's mood toward social, political, and economical issues [9, 128]. For instance, O'Connor et al. [128] aligned public opinions extracted from traditional polls with sentiments measured from Twitter. The authors detected a high correlation rate between sentiment word frequencies in Twitter messages and consumer confidence levels and political opinions as indicated by the polls. Similarly, Bollen et al. [16] analyzed the textual content of daily Twitter feeds by mood tracking tools. The authors reported that the accuracy of stock market predictions can be significantly improved by the inclusion of specific Twitter mood dimensions. Following this line of research, in this chapter we assume that emotions expressed by end users of software in software systems' Twitter feeds represent a valuable source of information for software developers. Such information can be leveraged to understand users' reaction to newly released software. An underlying tenet is that user involvement in the software process is a major contributing factor to software success [11].

In what follows, we demonstrate through the new operating system iOS10's Twitter feed how the public opinion, as measured by Twitter sentiment, fluctuates in correspondence to specific software-related events.

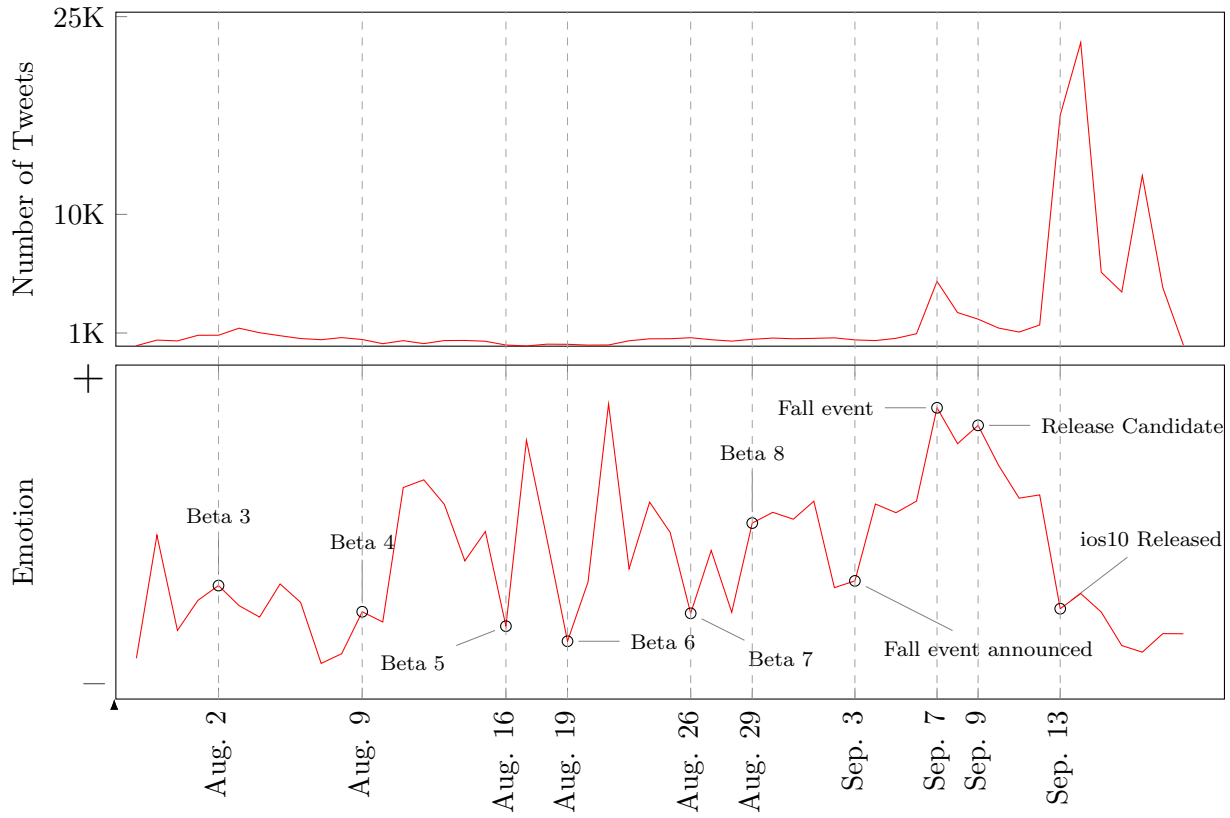

Figure 3.1. Number of tweets per day (top), and aggregate sentiment polarity (bottom) with important events marked.

3.2.1. Example: The *IOS10* release

On September 17th, 2016, Apple announced their new operating system for their popular smart phone the iPhone. We collected tweets mentioning the term "ios10" or containing the hashtag "#ios10" over the period from July 29th to September 19th. We analyzed the sentiment in each tweet using *SentiStrength*, a tool that associates common words and phrases with sentiment scores[1]. SentiStrength returns an integer value in the range [-4, 4], representing the overall sentiment of the tweet such that 0 is the neutral state. Figure 3.1 shows the average sentiment polarity calculated for our tweets over the data collection period, the total number of tweets collected, and important events related to iOS10.

During the public beta-testing process, tweets covered a variety of topics. For instance, the look-and-feel features of the new iOS has generated mainly positive reactions, with

1. `http://sentistrength.wlv.ac.uk/`

tweets such as "Ios 10 seems pretty smooth and have some beautiful animation. Im waiting. #iphone #iOS10 A whole new ios", and "I ain't gonna lie this #ios10 on my iPhone 6s is pretty aesteticly pleasing". The usability of the new OS, however, has generated some negative reactions, mainly due to users struggling with the new interface. For instance, an issue that frequently stood out in the tweets was user complaining about the loss of the swipe-to-unlock feature with tweets such as "won't upgrade to #iOS10 because the slide to unlock has been removed and I don't care any new features. #Apple #SlideToUnlock". In fact, the public negative reaction to this particular feature was so significant that it received news coverage by *CNN*:

> "to unlock an iPhone, you no longer swipe left but hit the home button. If you use a fingerprint to unlock every time, you won't notice. But others will repeatedly try to log on and instead see a new screen with weather, calendar and other bites of information. It will grow on us, probably."

Another controversial issue leading up to iOS10's release was the replacement of the default set of *emojis* with a new one. A number of tweets were posted on this topic, such as "I don't like ios10 emojis. they look like android emojis" and "I hate the emojis on ios10 so much how do I downdate". Our analysis also shows a spike in the positive sentiment on September 7^{th}, which is the date of Apple's Fall event (an annual keynote event). During this event the iPhone 7 was announced and a number of Twitter posts exhibited excitement and anticipation, in many cases directly referencing #AppleEvent. Tweets such as "Time to update my iphone #iOS10 #AppleEvent", and "I just can't wait to upgrade to IOS10 on Tuesday", were common.

A noticeable drop in sentiment corresponded with the actual release of iOS10, where the excitement was tempered by some frustration over technical issues. The rate of posting surged to over twenty thousand tweets the day of iOS10's release, and a number of users reported having their phone rendered unbootable (*"bricked"*) in tweets such as "iOS 10 over-the-air update bricked my 6s. Downloaded, installed, then rebooted to a plug in to

iTunes notice. iTunes has to fix. #ios10" and "Bricked my iPhone while updating to iOS10, stuck in recovery mode. Restoring in iTunes now". We also see a public reaction to changes introduced in the beta, but were only experienced by many users on release. Ordinary users reacted more harshly to the new emojis, echoing the sentiment expressed during the early betas: "Main reason I havent́ updated to iOS10 yet is simply because of the gun emoji...that really just makes me mad", and "i hate how emojis look on ios10 lmao".

3.2.2. Motivation

The iOS10 example shows how public sentiment, as measured by SentiStrength, can drastically change in response to specific software-related events. However, like many sentiment analysis tools, SentiStrength adheres to a uni-dimensional model of mood, making binary distinctions between positive and negative sentiment. Naturally, in our analysis, we assume that a negative sentiment is associated with a bad experience, such as a buggy update or a disappointing beta. A positive sentiment, on the other hand, might indicate a positive experience, such as a new feature being well-received. This binary classification of sentiment, while possibly giving a generalized indication of the public sentiment, may ignore the rich multi-dimensional structure of the human mood. In particular, the human positive and negative moods can be further broken down into specific emotions such as anger, excitement, and frustration. Each of these emotions conveys a different type of information that can be interpreted in various ways. For example, in their analysis of the stock market's movement, Bollen et al. [16] observed that public mood states measured in terms of positive vs. negative did not result in any correlation with stock market movement; rather, the specific emotional dimension *"calm"* was the most predictive.

Motivated by these observations, in what follows, we investigate the performance of various sentiment analysis techniques in detecting emotion information expressed in software user tweets. Our ultimate goal is to build a sentiment analyzer that is customized to detect and translate these emotions into actionable software engineering requests.

Table 3.1. Sample software-relevant emotions

Word/phrase	Polarity	Emotion	Examples
crash, not-working, fix	negative	bug report	"@VisualStudio Visual Studio15 Preview new installer not working"
listen, waste-of-time	negative	frustrated with update	"Listen to the people yik yak... fix it :(#yikyak"
uninstall, bring-the-old, go-back, change-back, ruin	negative	unsatisfied with update	"Can we bring the old #facebook back" "Well @PokemonGoApp new tracking ruined the game for those not living in the city. Nice way to kill the game. #uninstall"
re-download, addicted, obsessed	positive	satisfaction	"to say that im obsessed with the new #snapchat filters is an understatement" "finally!!! I can now redownload #yikyak back"
cannot-wait, excited-for	positive	excitements and anticipation	"can the new #callofduty come out already? cant wait ugh"

3.3. Analysis and Approach

This section describes our data collection, qualitative analysis, and automated classification process, and presents and discusses our results.

3.3.1. Data Collection

We used the Twitter *Search API* to collect our dataset. This API takes a search query (a string, which may contain hashtags, plain text, and mentions) related to a certain topic, and returns a set of tweets matching the query (i.e, potentially relevant to the search topic). To conduct our analysis, we collected tweets from the Twitter feeds of 10 software systems, sampled from a broad range of application domains. These systems include: *Windows10, Android, AppleSupport, CallofDuty, Chrome, Instagram, Minecraft, Snapchat, VisualStudio,* and *WhatsApp*. We limited our data collection process to tweets

addressed directly to the Twitter account of a given software product (e.g., tweets including *@Windows10*). This strategy ensures that only tweets meant to be a direct interaction with the software provider are included. The data collection process was repeated on a daily basis from April 6^{th} to June 4^{th} of 2016, with duplicate tweets being discarded. The resulting dataset held $360,873$ tweets.

3.3.2. Qualitative Analysis

To create our ground-truth dataset, 1000 tweets were randomly sampled from our dataset. These tweets were manually examined by two human annotators, with an average 5 years of experience in programming, to identify subjective expressions. A subjective expression is any word or phrase that is used to express an opinion, emotion, evaluation, stance, or speculation. Each tweet is classified at two levels of abstraction, including its general emotional polarity (positive, negative, and neutral) and the specific emotions it carries (sub-categories of the general negative or positive emotion the tweet conveys). Our qualitative analysis revealed the following types of emotions in our collected tweets:

- **Frustration**: A frustration feeling typically signifies the presence of bugs or unwanted behavior (e.g., "@ifunny im sick of random ads poping up everywhere"). For instance, in the domain of video games, *frustration* often signifies excessive difficulty or problems in game control (e.g., "@CallofDuty @Treyarch Any chance you can make spawns worse? I don't feel like I've had a proper game if Im not spawn killed 8 times a match". Frustration is typically associated with terms and phrases such as *sick*, *kill*, and *frustrating*.

- **Anticipation and Excitement**: Tweets with high anticipation inform developers about the features users are looking forward to, for example "@googlechrome I'm starting to regain my respect for Chromebooks. Because you are putting Google Play. Man i'm so hyped.". Excitement commonly appears in video game Twitter feeds, for example, a user bringing attention to their achievement in a game (e.g.,

"@Minecraft Check out this @Sway I made! "hero of the city"""). Anticipation and excitement are typically associated with terms and phrases such as *hyped*, *looking forward to*, and *can't wait*.

- **Satisfaction**: Satisfaction and dissatisfaction emotions are typically related to how users feel about software features. For example, when a new feature is well received, users often react with tweets such as "@googlechrome love the canary build of chrome i thought it be alot buggy but its not its working fine and like the new look its alot better". When features are poorly received, users often react with tweets such as "@instagram I'm sorry but I absolutely HATE the new update". Users' level of satisfaction allows developers to plan better for future patches and updates. Satisfaction is associated with phrases such as "love the new" and "great job on", while dissatisfaction is typically found with phrases such as "change it back" and "why did you".

- **Bug reports**: A bug report is not an emotion *per se*, however, such tweets often carry a compound negative sentiment. More specifically, tweets reporting problems are often accompanied by frustration and dissatisfaction feelings, expressed through phrases such as "please fix" and "any solutions?" (e.g., "@googlechrome Your android upgrade removed all of my open tabs. Going thru history to get them back is onerous can you fix this?").

A summary of our detected emotions as well as the list of their evoking expressions and sample tweets is shown in Table 3.1. Table 3.2 shows the number of tweets in our dataset that exhibit the different specific emotions. The intensity of the emotion is not considered at this stage of our analysis.

3.3.3. Automated Sentiment Analysis

Manually filtering through massive amounts of tweets can be a laborious and error-prone task. Therefore, for any solution to be practical, automated support is needed to

facilitate a more effective data filtering process that can capture, with a decent level of accuracy, emotions in software-relevant tweets.

In general, automated sentiment classification methods can be classified into unsupervised and supervised. The unsupervised approach relies on the presence of opinion lexicons, or emotion-indicator words, to estimate the sentiment polarity of the tweet based on the positive-to-negative word ratio, or simply the raw counts of opinion words [128]. The lexical approach focuses on building dictionaries of labeled words, where each word is given a score that indicates its emotional polarity. A common way to classify a text using these scores is by adding the positive values and subtracting the negative values of the terms in the text. If the total score is positive, the text is classified as positive, otherwise it is negative.

While the unsupervised approach can be easily implemented, it can be difficult to collect and maintain a universal sentiment lexicon as different words may reflect different meanings in different contexts [77, 137, 157]. Furthermore, simply relying on the presence of certain emotion words can lead to misleading results. This problem is often observed in Twitter messages due to their limited, and often ambiguous, textual content. For example, the tweet "@Microsoft I love #Windows10, but not as much as I loved #Windows8.1" carry both positive and negative feelings toward Windows 10. However, a dictionary-based sentiment analyzer will classify this tweet as positive due to the presence of the word *love* twice in the tweet.

The supervised approach, on the other hand, attempts to overcome these limitations by training prediction models (e.g., Naive Bayes and Support Vector Machines) on manually labeled tweets to make sentiment predictions for new data [32, 163]. OpinionFinder[2], is an example of a supervised sentiment classifiers that is trained on the Multi-perspective Question Answering (MPQA) Opinion Corpus. This corpus contains news articles from various news sources manually annotated for opinions (i.e., beliefs, emotions, sentiments,

2. http://mpqa.cs.pitt.edu/opinionfinder/

Table 3.2. Number of tweets conveying each emotion

Emotion	**#Present**	**#Absent**
Frustration	209	284
Dissatisfaction	133	360
Bug Report	218	275
Total negative:	493	
Satisfaction	182	177
Anticipation	42	317
Excitement	131	228
Total positive:	359	

speculations, etc.). A main limitation of this approach is that a model that is trained using a certain corpus might not be able generalize well for other domains. Furthermore, preparing large enough datasets of manually labeled emotion text is a labor-intensive, extremely subjective, and time-consuming task [99].

To classify our data, we investigate the performance of two text classification algorithms, including Naive Bayes (NB) and Support Vector Machines (SVM). These two algorithms have been heavily used in Twitter sentiment analysis and have shown interchangeably good performance across a broad range of tasks [13, 163]. To implement NB and SVM, we use Weka[3], a data mining software suite that implements a wide variety of machine learning and classification techniques. SVM is invoked through Weka's Sequential Minimal Optimization (SMO) class, which implements John Platt's algorithm for training a support vector classifier [134]. In our preliminary analysis, we find the default linear kernel of SVM to be most effective for Twitter sentiment classification. To train our classifiers, we use 10-fold cross validation. This technique creates 10 partitions of the dataset such that each partition has 90% of the instances as a training set and 10% as an evaluation set.

3. http://www.cs.waikato.ac.nz/~ml/weka/

Table 3.3. Polar Sentiment Classification Results

Classifier/Dataset	Precision	Recall	F
SentiStrength Positive	0.76	0.73	0.74
SentiStrength Negative	0.69	0.65	0.67
NB Positive	0.81	0.81	0.81
NB Negative	0.77	0.77	0.77
SVM Positive	0.78	0.78	0.78
SVM Negative	0.70	0.70	0.70

3.3.4. Results and Discussion

To get a sense of the performance of NB and SVM, we initially classify the data using SentiStrength. In our analysis, we classify a tweet as negative if it gets a score of -2 or less and as positive if it gets a sentiment score of +2 or more. Table 3.3 summarizes the classifiers' performance. The relatively weak performance of SentiStrength in comparison to the supervised methods can be attributed to the lack of software-specific words in its sentiment dictionary. For example, the tweet "@Windows > @apple The surface is smaller but yet more powerful! #WindowsVsApple @surface" has positive polarity. SentiStrength misclassified this tweet as negative due to the presence of the word *smaller*. In English, this word leans toward negative sentiment, but in the context of software, smaller size is typically a positive trait. Along these lines, the tweet "@VisualStudio is VSTS going SUPER slow?" was miss-classified by SentiStrength as positive due to the word *super*.

In comparison, our results show that the supervised classifiers NB and SVM managed to achieve decent levels of accuracy in detecting general (Table 3.3) and specific emotions (Table 3.4), with NB outperforming SVM. In general, both classifiers recognized many positive and negative emotion-evoking software-specific words that SentiStrength missed. For example, the words *challenge* and *backwards* are typically associated with negative sentiment in day-to-day English. However, in the context of video games, the word *challenge* typically indicates a good mood. The word *backwards* often appears in the context

Table 3.4. Specific Emotion Classification Results (NB and SVM)

	NB			SVM		
Emotion	**P**	**R**	**F**	**P**	**R**	**F**
Frustration	0.71	0.71	0.71	0.68	0.68	0.68
Dissatisfaction	0.75	0.77	0.76	0.77	0.77	0.77
Bug Report	0.75	0.75	0.75	0.71	0.71	0.71
Satisfaction	0.75	0.76	0.75	0.64	0.64	0.64
Anticipation	0.86	0.89	0.87	0.85	0.87	0.86
Excitement	0.75	0.76	0.75	0.85	0.85	0.85

of backwards compatibility, which is also a positive attribute of software. Similarly, the supervised methods managed to capture words which are considered neutral in day-to-day English, but which are associated with negative sentiment in software, such as *uninstall* and *rollback*. For example, both NB and SVM detected that the word *please* is almost always an indication of negative sentiment. In general, the word *please* comes associated with requests to help with software problems, such as "@googlechrome please i need ur help! i don't have sound with google chrome!! and i have not found a solution yet!! could you help me please". Similarly, tweets with the words *fix*, *bug*, and *crash*, were classified as negative as they are typically associated with requests for bug fixes and systems crashing.

In terms of general emotional polarity, our results show that the supervised classifiers can be effective in detecting software-specific emotion words that are often missed by general-purpose sentiment analyzers. However, getting such techniques to achieve acceptable accuracy requires preparing large-scale datasets of manually annotated Twitter data. This can be a challenging task as the language of Twitter messages evolve at very fast pace and keeping track of such noisy colloquial terminologies can be a time-consuming task.

3.4. Conclusions and Future Work

The research on mining micro-blogging services for software engineering purposes has focused on the way developers use such platforms to share and exchange software development information. Analysis of sample software-developers' tweets has revealed that

they frequently make use of social media as a means to facilitate team coordination, learn about new technologies, and stay in touch with the interests and opinions of all stakeholders [142, 155]. In our analysis, we shift the attention to software users' tweets rather than developers' tweets. In particular, we show that emotions expressed in software systems' Twitter feeds can be a non-traditional source of feedback that enables software developers to instantly connect to, interpret and rationalize their end-users reactions.

Our analysis is conducted using 1000 tweets sampled from the Twitter feeds of multiple software systems. A manual qualitative analysis was conducted to classify these tweets based on their sentiment polarity and specific emotions they express. The data was then automatically classified using SentiStrength and two general purpose text classifiers, including NB and SVM. The results showed that NB and SVM were more accurate than SentiStrength in detecting the emotional polarity of software tweets. Furthermore, both classifiers were able to capture the fine-grained dimensions of human emotions with decent levels of accuracy.

The work presented in this chapter can be described as a preliminary proof-of-concept analysis of the value of emotion information in software-relevant tweets. The proposed work can be extended along two main directions, including:

- Analysis: A major part of our future analysis will be focused on improving the accuracy of the sentiment classification model. For instance, in our future work, non-word sentiment signals in tweets, including emoticons and punctuation, will be considered classification features. Emoticons can be used to predict the main sentiment of the tweet and the intensity of the emotion [162]. For example, a sad smiley `:(` indicates a negative feeling, while the `:(((` emoticon expresses a more intense sense of dissatisfaction [99]. Basic English punctuation can also be used to measure the intensity of the emotion. For instance, multiple question marks often indicate more intense confusion, while multiple exclamation points indicate intensity on both the positive and negative side. Furthermore, the analysis in our chapter is limited to

emotion-evoking uni-grams, or words. Related research has shown that more complex emotions are better expressed through expressions and phrases (combinations of uni-grams and bi-grams) [46, 130]. Therefore, part of our future work will be to examine emotion-evoking phrases and word structures (n-grams) in software-relevant tweets.

- Applications: One of the main challenges in our proposed work is to develop a mechanism for automatically interpreting various types of emotions in different application domains. Our goal is to devise an emotion-aware model that can provide instant recommendations to software developers based on the public's mood.

Chapter 4. Modeling Crowd Feedback in the App Store

Recent analysis of mobile application (app) stores has shown that apps are subject to steep download distributions, where a handful of successful actors take most of the revenue. In this winner-take-all environment, it is critical for developers to take advantage of competitors' failures—particularly when apps are abandoned by users—in order to survive in such a highly competitive and volatile market. To shed more light on this phenomenon, in this paper, we present a case study on the rise and fall of Yik Yak, one of the most popular social networking apps at its peak. In particular, we identify and analyze the design decisions that led to the downfall of Yik Yak, track other competing apps' attempts to take advantage of this failure, and perform user-driven feature-oriented domain analysis to understand and model users' concerns within the domain of anonymous social networking apps. The objective of the proposed analysis is to provide systematic guidelines for tracking and modeling the shift in app users' requirements and expectations and predicting the impact of feature updates on app user acquisition and retention rates.

4.1. Introduction

Analysis of large datasets of app store reviews has revealed that almost one third of these reviews contain useful information (e.g., bug reports and feature requests) [27, 106, 112]. Social media platforms, such as Twitter, have also been exploited as a more open, widespread, and instant source of technical user feedback. Recent research, along with the work expressed in this dissertation, has shown that almost 50% of software-relevant tweets contain useful information for developers [56, 57, 165]. However, despite this effort, there is still a research gap on how such information can be systematically utilized to enhance app survivability and competitiveness.

In an attempt to bridge this gap, in this chapter, we present a case study on the rise and fall of the social networking app Yik Yak. Between 2013 and 2016, Yik Yak managed to grab a massive share of the social networking market by offering a unique combination of features that attracted users' attention. However, following a series of feature updates, in mid-2017,

the development team of Yik Yak announced that the app would be shutting down due to a massive loss in their user base. The story of Yik Yak presents a unique opportunity for analyzing one of the most recent major failures in the app market. *Learning from failure* has long been used in market research as a means to build organizational knowledge and prevent future failures [89, 108].

To conduct our analysis, we collect and synthesize user feedback available on app stores, social media, and several news outlets to track the story of Yik Yak. We examine its most successful features, the design decisions that led to its decay, and eventually, death. We further track user migration patterns to other competing apps and analyze the main concerns of users of these apps. The main objective of our analysis is to demonstrate the feasibility of systematically generating unified domain models that can integrate multiple heterogeneous sources of user, market, and system information to reflect an accurate picture of the current state of the domain and its most desirable features.

The remainder of this chapter is organized as follows. In Section 4.2, we track the history of Yik Yak and identify its main rival apps. In Section 4.3, we perform an in-depth feature-oriented analysis of the domain of anonymous social networking apps. Section 4.4 discusses the main findings of our analysis and their potential practical significance. Section 4.5 reviews seminal related work. Finally, in Section 4.6, we conclude the chapter and outline prospects of future work.

4.2. Yik Yak's History and Market Analysis

In this section, we track the history of Yik Yak, including its main features, most notable updates, decline, and eventually death. We further perform market analysis to identify the set of rival apps that former Yik Yak users have migrated to.

4.2.1. The Rise and Fall of Yik Yak

Yik Yak was a location-based social networking app that allowed users to post and vote on short messages (known as yaks). Yik Yak distinguished itself from its competitors (e.g., Twitter, Facebook, and Snapchat) by two main features: anonymous communication

and geographical locality. In particular, users could only post and engage in discussions within a 1.5 mile radius around their location. This feature ensured that most of the posts were related to activities in a local area (e.g., a college campus or a small town). Yaks were sorted primarily by distance, so that posts about particular buildings or activities on a campus would find the most relevant local audience. In addition, users had the option to set a home community, called a *herd*. Herds allowed a user to communicate with friends even if the user was temporarily outside of his/her original geographical area.

One of Yik Yak's most notable features was anonymity: users did not need to register a name in order to participate in any of the app's features, including creating threads, posting replies, and voting on messages. This feature was widely praised, with news outlets such as Wired stating,

> From my interactions on Yik Yak, I can tell how well the buses are running on a given day, the least crowded times at the gym, and student reactions to administrative decisions. Instead of being worried about the negative aspects of anonymity on Yik Yak, we should embrace the service's ability to help our young people grow and develop a strong sense of community. [76]

The combination of anonymity with locality proved successful; at the end of 2014, Yik Yak had become one of the most popular social networking platforms on college campuses. In fact, due to this massive success, in 2015, the company was valued at close to $400 million. Business insider writes,

> Goetz ended up leading a $62 million Series B round in Yik Yak at a valuation that approaches $400 million. Goetz admits the app's current scale may not seem to justify $400 million, but he says its engagement was enthralling. [140]

The anonymity provided by Yik Yak had a downside, however. Due to the lack of personal identifiability, Yik Yak posts were an easy vector for cyber-bullies to anonymously harass other users at their schools or universities. This problem became significant when the

app was used to make threats credible enough for institutions to request police assistance. The New York Times reported,

> Local police traced the source of a yak — 'I'm gonna [gun emoji] the school at 12:15 p.m. today' — to a dorm room at Michigan State University. The author, Matthew Mullen, a freshman, was arrested within two hours and pleaded guilty to making a false report or terrorist threat. [109]

In addition to threats, Yik Yak became a platform for spreading toxic behavior in high schools, causing a *change.org* petition for shutting down the app to receive more than 78,000 signatures. In an attempt to minimize the frequency of such incidents, the app's founders implemented *geo-fences*, which detected when the app was being used from within a high school and prevented access. About this measure, Business Insider writes,

> they proactively blocked about 100,000 middle schools and high schools from accessing the app to keep under-agers off Yik Yak. They also implemented community policing tools and keyword targeting for hate speech. [141]

While Geo-fences significantly restricted negative behavior in middle and high schools, other measures were still necessary to prevent toxicity elsewhere. Following this effort, Yik Yak began to roll out mandatory *handles*, where users would be forced to choose a screen name. At first, the option to hide these handles allowed users to continue posting completely anonymously, but with the release of version 4.0.0, even this ability was removed. The response from the community was swift and overwhelmingly negative, with tweets such as "Rip yikyak- was only good when it was anonymous" becoming more common on the app's Twitter feed. App store reviews were similarly hostile, with reviews such as "Anonymity is the sole reason for which I downloaded this app in the first place. Handles just ruins it!" and "I see no point of using this app if its going to be no longer anonymous!" began to dominate the reviews.

This series of feature updates, along with other updates (e.g., removing the herd feature), had catastrophic consequences on Yik Yak's market performance. The number of

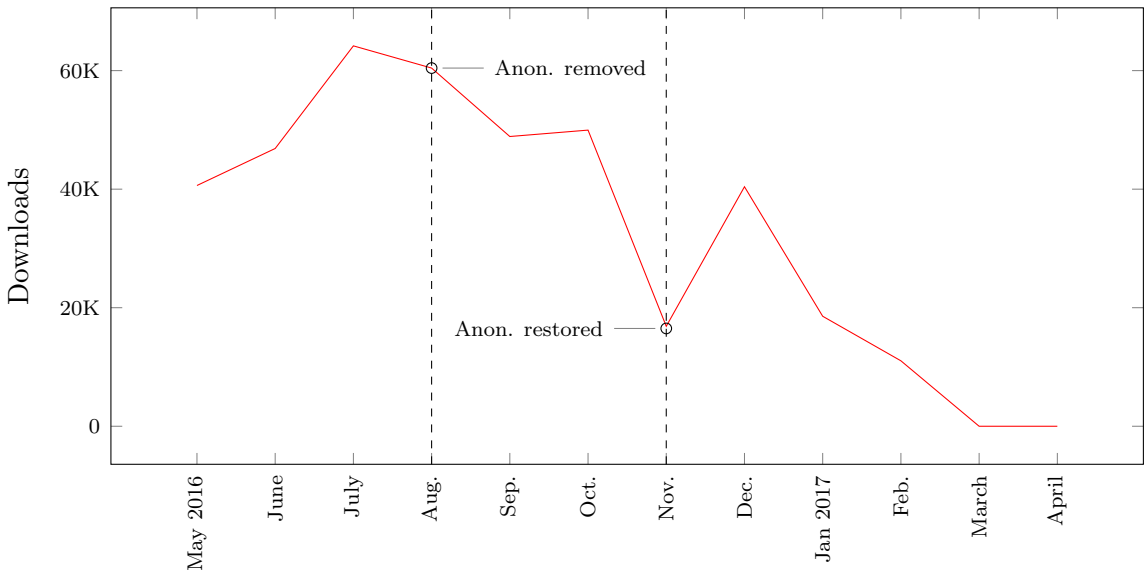

Figure 4.1. The number of monthly downloads after anonymity (Anon.) was removed and then restored in Yik Yak.

downloads (as estimated by prioridata[1]) sharply dropped as shown in Fig. 4.1. Yik Yak's attempt to reverse course by reintroducing anonymous posting, was never successful in recapturing its previous popularity. This massive loss of users, besides other miss-calculated business decisions, led the company to officially suspend operations in May of 2017. Eventually, the company was acquired for a fraction of its original valuation. On that matter, USA Today writes,

> The Atlanta-based company had been for sale, but apparently had few takers. It was able to move some of its engineers to Square, the online-payments app, for what Bloomberg says was $3 million. [51]

4.2.2. User Migration

Yik Yak offered a unique form of anonymous, locally-focused, and democratic communication. After the app was suspended, several competitor apps emerged to take advantage of the demand for a replacement. Numerous threads on social media appeared with former users discussing potential alternatives.

1. `https://prioridata.com/apps/yik-yak-com.yik.yak/performance`

In order to understand Yik Yak users' migration behavior, we identified the set of popular apps that attempted to imitate Yik Yak's feature set. We looked especially for apps which users on Reddit, Twitter, and app stores explicitly stated that they were switching to as a result of Yik Yak's suspension. Gathering and synthesizing knowledge from mainly user-driven sources can help to narrow down the domain to its most fit elements from a user perspective, thus saving the unnecessary effort of analyzing hundreds of apps that, even though share the same core functionality, failed to survive or grab any user attention (i.e., eliminated by market selection).

Recent analysis has revealed that users typically discuss competing apps in their app store reviews [83]. Such information can be leveraged to identify other apps that can be potentially classified under the same domain. For example, after Yik Yak's failure to control for cyber-bullying, names of rival apps that implement a better content moderation started appearing in reviews of the type "I'm moving to [Spout], its less infested with trolls and racists and it has a more mature and interesting user base." and "I will be switching to [Swiflie] after Yik Yak shutdown.". Furthermore, people often resort to social media (e.g., Twitter) to express concerns about their apps, such as discussing alternative apps and providing recommendations to other users about potential competing apps to migrate to. For instance, after Yik Yak's suspension, tweets of the nature "Any alternatives to #YikYak?" and "@YikYak we cant lose our community where should we go now" started appearing on Twitter. Other social media outlets, such as the social media aggregator Reddit, also commonly include discussion threads about app alternatives. For example, one user posted a Reddit thread titled "Alternative to Yik Yak?"[2] trying to convince other users to try the now defunct competitor app *Candid*. Part of the post reads,

> As for Candid it's much better IMHO. You can share links, not have to be on location at a college to post in the college's feed, you can share post links and edit posts and comments. Most importantly it's 100% anonymous. So try it out if you're interested!.

2. https://www.reddit.com/r/yikyak/comments/4y5lpl/alternative_to_yik_yak/

Comments on that post also expressed support for other alternative apps, such as Swiflie, Jodel, and Spout. Similar recommendations were also made in several other online news outlets, such as the news articles "Five Best Yik Yak Alternatives You Should Check Out" and "Stay Anonymous Online With the 5 Hottest Yik Yak Alternatives". Names of competitor apps have also appeared on *Alternative To*, a crowd-sourced website for submitting and discovering alternatives to software applications.

Our effort to identify Yik Yak's competing apps included manually analyzing Yik Yak related tweets (any tweets including *#YikYak* or *@YikYak*) and reviews that appeared in May and June of 2017 along with any Reddit or news posts that discussed the dying app. Specifically, we kept track of any names of potential competing apps that users expressed interest in. Furthermore, we only considered apps with an average of 1,000 downloads over the months of May and June of 2017. This number is essentially a lower bound. More specifically, it is difficult to imagine a social network with fewer than 1,000 monthly downloads achieving coverage over major universities and metropolitan zones. Table 4.1 summarizes the number of global downloads of popular alternative apps around the time of Yik Yak's death. These numbers were obtained from Prioridata, a web service which provides app download estimates based on the freely-available global download rankings from both app stores. While other factors, such as user retention and usage are also necessary to accurately quantify popularity [88], the number of downloads alone can be a valid proxy, or at least an indication, of app popularity [133]. In what follows, we describe the main alternative apps identified in our analysis along with their most popular features:

- **Kik:** Kik is a social media app which actually predates Yik Yak by four years, with the first tracked version being released in 2010. Unlike Yik Yak, Kik has never had a locality requirement: messages can be viewed by any member of a group no matter how far away they are. Additionally, it has never allowed truly anonymous communication, meaning that there was never a user outcry against such a feature being removed. Based on download numbers alone, Kik is clearly the most popular

Table 4.1. Estimated combined number of global downloads (Google Play and the Apple App Store) during May and June of 2017 for each of Yik Yak's rival apps.

App	May	June	Change
Kik	1,672,000	1,695,200	1.4%
Jodel	262,300	200,400	-23.6%
Firechat	41,100	42,700	3.8%
Whisper	142,800	148,400	3.9%
Swiflie	26,468	10,100	-61.8%
Spout	3390	515	-84.8%

alternative with over one million downloads per month. However, this popularity has a downside. Whereas Yik Yak struggled to cope with cyber-bullying, Kik has been tainted by numerous, serious incidences of being used for kidnapping, child exploitation, and murder. The New York Times writes about one such incident,

> The death of Nicole Madison Lovell, a liver transplant and cancer survivor from Blacksburg, Va., has put Kik in the spotlight. Neighbors say that the day before she died, Nicole showed them Kik messages she had exchanged with an 18-year-old man she was to meet that night. [149]

- **Jodel:** Launched at almost the same time as Yik Yak, Jodel targeted a European userbase. After Yik Yak's poorly-received 4.0.0 release, users from English speaking countries began migrating to Jodel. Jodel's developers noticed this trend and reached out to Reddit's Yik Yak community for suggestions to improve the experience for former *Yakkers*. Among the most popular suggestions were access to herds (now called *hometowns* in Jodel), better moderation for anti-bullying and threat purposes, and improved notifications. All three suggestions were implemented to varying degrees, enabling Jodel to emerge as a popular alternative to Yik Yak.

- **Firechat:** Firechat is an unconventional Yik Yak alternative that utilizes mesh networking to allow local, anonymous discussions. Unlike other social networking apps,

Firechat connects users in-range directly to one another's phones rather than sending information over the Internet. Users' posts propagate by being encrypted and re-sent across nearby devices running Firechat until no more users can be reached. Posts can be made to anyone in the local area, or pre-filtered into focused chat groups. Firechat's direct messaging system makes it usable when cell or data connections are unreliable or unavailable. Business Insider writes,

> FireChat became hugely popular in Iraq after the country faced restrictions on internet use, and it was an integral part of the 2014 Hong Kong protests and the 2015 Bersih anti-corruption movement in Malaysia. [64]

- **Whisper:** Whisper is a unique app which allows users to post anonymous thoughts and confessions they may be reluctant to share with their identity attached. Unlike its competitors, Whisper is not designed around comment threads, where users reply to each other in sequence. Instead, posts can have many replies in parallel, and each reply can have its own set of replies, often forming a vast tree containing hundreds of posts that spiral into new discussions. This lack of direct discussion threads makes intimate conversations unwieldy, nonetheless, it facilitates the sharing of a wide variety of opinions on a single topic. According to Wired magazine, this organization of discussion may reduce toxic behavior,

> Whisper aspires to be a social network that doesn't rely on a social graph. The app has features for finding people nearby or forming groups, but the feed defaults to showing users popular posts from all over the world. The result is far less deeply offensive material [79]

Besides a unique method of conversation, Whisper also offers features designed to encourage local communication. Users' locations are tracked, and the posts in a user's area can trigger notifications indicating that a local discussion is taking place.

- **Swiflie:** Swiflie is a recent social networking app that was created specifically to provide an alternative to Yik Yak after the latter lost the ability to post anonymously. Swiflie adopted the tagline "Be yourself, or not" to market this fact. Optional profiles are also allowed, along with the ability to post to user groups, which replaces Yik Yak's herd functionality. In a major departure from Yik Yak, users are allowed to view and post on chat groups outside of their local area. Such groups exist for many large universities, metropolitan areas, and even social groups. Swiflie is specifically designed to encourage timely posts, and every post is eventually removed. Posts normally last for about one day, but this lifespan can be extended if a post receives enough *likes* from fellow users.

- **Spout:** Spout was released in October 2016, shortly after the 4.0.0 release of Yik Yak. Spout offers the closest experience of Yik Yak among the various alternatives, with an almost exact duplication of the feature set. Despite this, there are a few differences, including the ability to see posts from various user-selectable distances and significantly more active moderation. The moderation is praised by users looking for less toxic communities to get involved with. For instance, one review states,

  > Great app, and a worthy alternative to Yik Yak. Like Yik yak, posts can be up-voted and downvoted, and you can also post images or GIFs in replies . . . Spout is less infested with trolls and racists than rival apps such as Candid and Swiflie, and it has a more mature and interesting user base as a result.

  However, many more users appear to regard the moderation as heavy-handed, with a large number of one-star reviews of users complaining about being banned. For example, one review states,

  > This app is very strict about banning people, if anyone doesn't like your post they have the option to report you which will accordingly ban you from using this app forever. I used this app for about a week and got banned quickly . . . The app has cool features that you can not find in (RIP) yikyak, but the banning is really bad.

Table 4.2. A feature matrix extracted from the domain of Yik Yak and its rival apps.

App	Loc.	Enc.	Prof.	Away	Grps.	Mod.	Sign	Ads	Anon.	Blk.	Not.
Kik			X	X	X	X	X		X	X	X
Jodel	X	X	X	X		X			X		X
Firechat	X	X	X	X	X		X	X	X	X	X
Whisper	X	X		X	X	X		X	X	X	X
Swiflie	X	X	X	X	X	X	X	X	X	X	X
Spout	X	X		X		X			X		X

4.3. Domain Analysis

In this section, we systematically analyze and model the main user concerns in the domain of Yik Yak and its competing apps along with their relations to the core features of the domain. In software engineering, feature-oriented analysis is often applied to describe the main reusable assets (e.g., requirements, design, and code) of Software Product Lines (SPLs). An SPL can be described as a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [135]. While reuse is not the main focus of this chapter, SPL analysis can be a powerful tool to obtain an in-depth look into the main features and user goals of our domain. Specifically, our analysis in this section can be described as a three-step procedure:

1. Extracting the core features of the domain

2. Extracting the main user concerns of the domain

3. Modeling the relations between user concerns and the domain features

In what follows, we describe these steps in greater detail.

4.3.1. Extracting Domain Features

We start our analysis by creating a feature matrix based on the observed behavior of each of Yik Yak's rivals identified earlier. A feature matrix describes the main features

of a group of systems in a certain domain. A feature can be defined as any observable functional behavior of the system. Each row of the matrix represents a feature vector, showing the features that each system in the domain supports [92, 93]. To create such a matrix, we downloaded and exercised the features of all apps in our domain. In particular, each author downloaded all of the apps on their personal smart devices, created an account for each app (if necessary), and then used each app multiple times over the period of two weeks to get a sense of the most dominant features they provide. Furthermore, the textual description available on the app store page of each app was used to capture the main features the developers of each app emphasize. A discussion session was then held to map our observations into a domain-specific feature matrix. Our analysis resulted in the feature matrix in Table 4.2. The following is a detailed description of these features.

- **Local posting (Loc.)**: apps supporting this feature use the location of posters to sort and filter posts for display, or alternatively, detect appropriate groups for users to join based on their location.

- **Encryption (Enc.)**: apps supporting this feature use some sort of data encryption to prevent outside actors from determining the content of an intercepted message. While encryption can give users a feeling of security in their anonymity, it can also prevent police from following up on crimes or threats of self-harm.

- **Profiles (Prof.)**: apps supporting this feature implement special pages for users to describe themselves. Profiles reveal when posts have a common author, which might hurt anonymity as active users can be identified based on their posts over time.

- **Away posting (Away.)**: the ability of users to post outside their local area.

- **Groups (Grps.)**: are chat rooms that allow users to post messages only to people interested in a particular topic, thus allowing more focused discussions.

- **Moderation (Mod.)**: is considered present if there are easily accessible mechanisms to report toxic behavior. Given the consistent cyber bullying and harassment that Yik Yak had suffered from, it is unsurprising that most of the apps in our domain have strong moderation.

- **Sign-ups (Sign.)**: refers to whether an app has a mandatory sign up before users can access its features. Given the difficulty of balancing anonymity and convenience with the need to provide effective moderation, it is unsurprising that there were several apps erring in each direction, either requiring users to sign up or not.

- **Advertising (Ads.)**: is a common monetization strategy for free apps. The specific implementation varies among apps. For example, Whisper shows ads as regular posts, while Kik allows users to chat with a virtual representative of the advertising entity.

- **Anonymity (Anon.)**: is a feature common to all apps in our domain. Apps with a mandatory sign-in feature associate posts with a persistent username, but none requires that these assumed names to be associated with real-world identities.

- **Blocking (Blk.)**: enables users to manually hide unwanted posts and disallow direct messages made by other users. Blocking allows users to act as their own moderators.

- **Notifications (Not.)**: allows social networking applications to inform users of new posts in their area or any new comments on their posts. All apps in our domain send notifications, but to different extents. For example, Kik and Spout use notifications to inform users of replies to their threads, while Firechat uses notifications to push advertisements on users.

4.3.2. User Concerns

Under this phase of our analysis, we are interested in users' concerns. A user concern can be defined as any direct or indirect, desirable or undesirable, behavior of the app that might result as a side-effect of deploying the app, or any of its features, in its environment.

For instance, users might perceive a video game as being *entertaining* or *boring*, or a social networking app as being *toxic* or *engaging*. The objective of our analysis is to extract and synthesize the main concerns of users in the domain of Yik Yak and its competitors.

We start our data collection process by extracting the most recent (i.e., from the date of data collection and going backward) reviews and tweets for each app. We only included posts that actually contained a legitimate user concern. Posts with no useful information were discarded (spam, praise, insults, etc.). Posts were manually classified in order of most to least recent until obtaining a total of 50 posts expressing actual user concerns from each type of post (50 most recent concerns from the App Store reviews, 50 from Google Play reviews, and 50 concerns from Twitter). This process was repeated for all apps in our domain. For Spout and Swiflie, we ended up with only 91 and 122 concerns in total before we ran out of reviews and tweets for these two apps. Each relevant post was examined by the authors to determine which concern was most appropriate for classification. To control for complexity, if multiple goals applied, the one mentioned more, or which seemed to be the primary concern of the user, was picked.

Our qualitative analysis shows that a total of seven concerns were present in the data. Examples of posts describing each of these concerns are shown in Table 4.3. Fig. 4.2 shows the total post count for each concern. In general, the set of concerns raised by the users in the domain of anonymous social networking apps can be described as follows:

- **Expressiveness:** refers to the ability of users to freely express themselves on the app. Many users react with hostility in their reviews and tweets to perceived breaches of their free speech, especially when it results in a ban.

- **Entertainment:** refers to how enjoyable the experience of using the app actually is. Entertainment has been identified as an important user requirement for social networking apps, often associated with the ability to communicate with a broad range of people [21, 98].

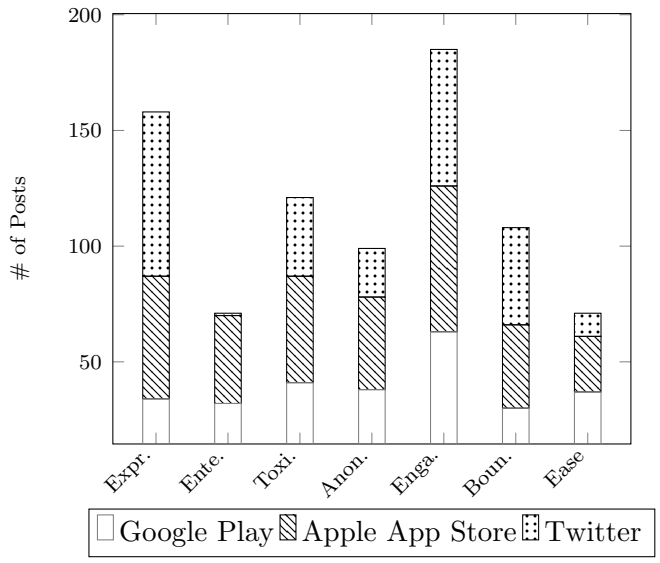

Figure 4.2. The number of posts (reviews and tweets) per category of user concerns.

Table 4.3. Example posts of the different user concerns.

Soft Goal	**Example Post**
Expressiveness	"I love the concept of expression through this app [Whisper]. Overall try it! Beware you might get hooked!"
Entertainment	"If you feel bored or just want to laugh, give it [Jodel] a try. Highly entertaining."
Toxicity	"The user base [on Spout] is extremely positive and supportive of each other, negative/bullying posts are downvoted quickly"
Anonymity	"I like the idea [of Jodel] and I think that the anonymous character is a lot [of] fun because people are discussing things they wouldn't do if the world knows who's behind the numbers"
Engagement	"After Tapt and Yak went down there was no where to go. Then Swiflie came along and brought us all back together; and in a whole new and better way."
Maintaining boundaries	"I like the concept of this app [Firechat]. The thing I don't like is that ANYONE can text you ANYONE any strangers can text you without your permission they should make it wear[sic] you have to request to message someone."
Ease of Use	"I hate that the video chat button is so easy to accidentally click [in Kik]. I've clicked it by mistake so many times during conversations."

- **Avoiding toxicity:** is the desire for a friendly environment for discussion. Toxicity is unavoidable on the internet, but effective moderation and reporting mechanisms can mitigate the extent to which users are exposed to such undesirable behavior.

- **Anonymity:** refers to users' ability to hide their identity when posting. Anonymity is highly desirable in our domain as it grants users the freedom to express their unpopular opinions. On the dark side, being anonymous makes it easier for users to behave in a toxic manner without jeopardizing friendships or social status.

- **Engagement:** refers to the ability of users to use the app in a way that affects their day-to-day life, especially regarding local posts, groups, and friends. In fact, making new contacts and engaging with existing friends have been shown to be the most important reason for people to use social networking apps [21].

- **Maintaining boundaries:** refers to the ability of users to control who talks to them and what posts are visible in their feeds. The presence of spam, excessive notifications, and annoying ads indicates the inability of users to set boundaries.

- **Ease of use:** is a common goal among almost all application domains. However, it meets our criteria of inclusion due to its particular importance in social networking apps [84]. Ease of use refers to the overall ability of users to access the app's features without frustration.

4.3.3. Modeling

In this section, we generate a unified user-driven feature model for the domain of Yik Yak and its competing apps (anonymous social networking apps). Several domain modeling techniques have been presented in the literature. One of the earliest frameworks proposed in the software engineering literature for modeling domain knowledge is Feature Oriented Domain Analysis (FODA) [78]. A feature model (FM) in FODA is represented as a hierarchical tree graph, showing the mandatory, alternative, and optional features of

the domain along with their commonalities and variabilities [120]. While FMs can be very effective for representing the functional features of the domain, they often ignore the non-functional requirements (NFRs). NFRs describe a set of quality attributes that a software system should exhibit, such as its usability, security, and reliability [45]. To represent such attributes, another type of models, known as the Softgoal Interdependency Graph (SIG), is used [49]. In SIG diagrams, softgoals are the basic units for representing NFRs. The interdependency relations in the graph are accompanied with plus and minus signs to indicate the type (trade-offs *vs.* synergy) and degree of impact between softgoals.

In our analysis, we adapt a hybrid domain modeling technique, known as Feature-Goal analysis (F-SIG), to describe the main goals and features of our domain of interest. F-SIGs enable a comprehensive qualitative reasoning about the complex interplay between the functional and non-functional features of a domain, allowing developers to record design rationale and evaluate different design choices early in the project [70]. In F-SIGs, a functional feature is represented using a rectangle shape, a softgoal is represented using a cloud shape. The edges of the graph represent the interrelationships among the softgoals and features of the domain. These relations are typically accompanied with plus and minus signs to indicate the type and degree of impact among softgoals and features (`--Breaks, -Hurts, ?Unknown, +Helps, ++Makes`). The F-SIG we derive to represent our domain is shown in Fig. 4.3.

The make/break associations expressed in the diagram are determined using the following coding scheme: posts (tweets and reviews) containing a specific user concern are grouped together. Each post in the group is then examined to determine which functional feature the user is concerned about (linking to the concern). These features are extracted and added to the graph. To determine the interrelationship (positive or negative) between a specific feature and the concern, we manually examine posts relating that feature to the concern. If more than 67% (two thirds) of these posts are leaning toward a certain sentiment polarity (+ *vs.* -), the relation is assigned to that polarity (`+Helps` *vs.* `-Hurts`).

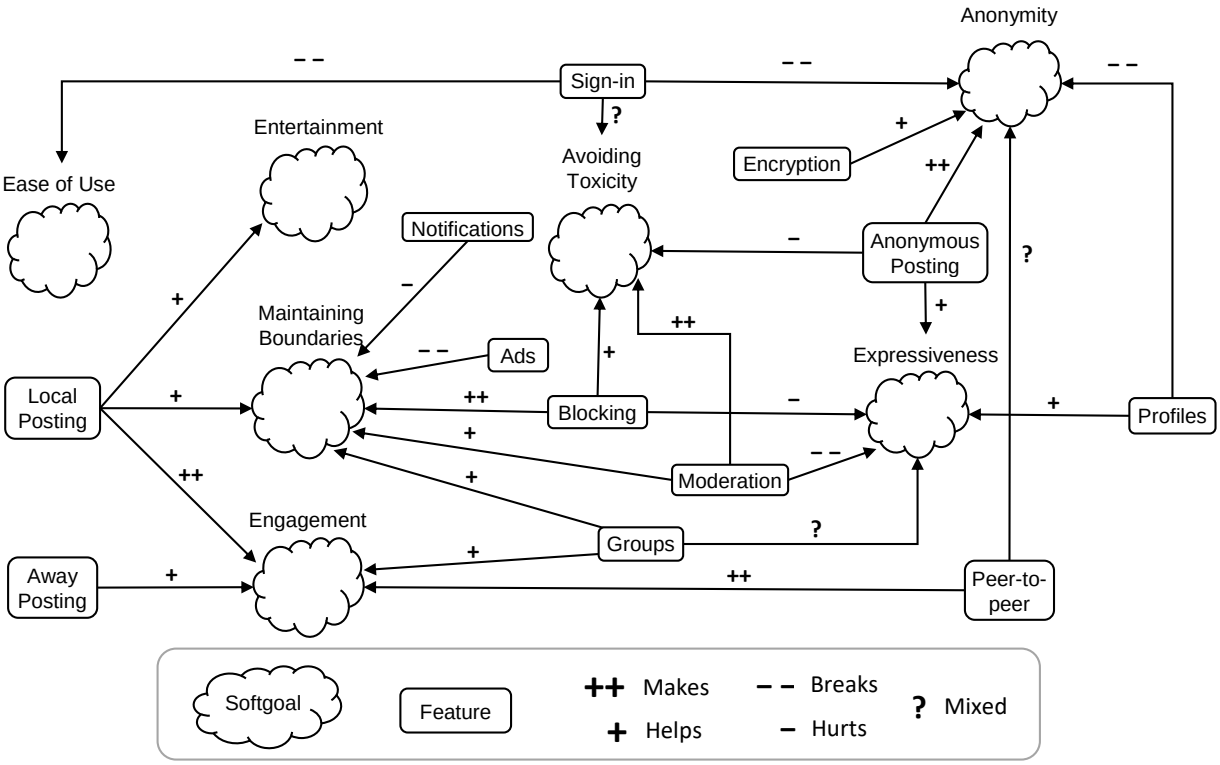

Figure 4.3. A model of the common user concerns and their relationships to the core features of the domain of anonymous social networking apps.

Otherwise, the relation is marked as `?unknown`. We then use the proportion of posts related to that specific feature to determine the relation intensity. In particular, assuming users have linked their concern to a number of N domain features, if the feature appears in more than $1/N$ of the posts, we mark that feature as having out-sized importance using `++Makes` and `--Breaks` instead of just `+Helps` and `-Hurts`.

For example, assuming 120 posts were found to be related to the concern *anonymity*. In their posts, users linked this concern to the features *sign-in*, *profiles*, and *encryption*. Assuming the majority of users perceive the *profiles* feature to hurt their anonymity (more than 67% of these posts related to having mandatory profiles are negative), we mark this relation as `-Hurts`. Assuming out of the 120 posts, more than $1/3 = 33\%$ are about profiles, we then upgrade the relation to `--Breaks`, thus indicating that this feature might have more severe negative impact on users' experience.

4.3.4. Model Interpenetration

Our coding procedure shows that *engagement* and *expressiveness* account for noticeably more user opinions. The most common form of *engagement* related posts expressed a user's desire to connect with lots of people. Thus, suggesting that a major factor for users to use a certain app is simply the ability to find friends on that app. Posts related to *Expressiveness* were mostly common on Twitter, primarily because many users were seeking to overturn a ban, often protesting that the terms of service were overly harsh on free speech. Twitter appears to be more popular than app store reviews for discussing bans, possibly because it is a low-latency method of communication with app developers. Many users also described problems with moderation, causing them to feel uncomfortable about *expressing* their opinions. For example,

> with the creators or mods controlling everything, I found that they ban people if they disagree with their opinion despite not being hateful and not for actual hate or racism.

Our analysis has also revealed that *toxicity* complaints were common. For instance, Spout seems to come closest to Yik Yak in terms of bullying behavior, with reviews like,

> Truly a place where you can see the ugliness of the internet and cyber bullying. The mob mentality of many users in this app makes it unappealing. It will never be better than yik yak.

Anonymity was also a concern for all users on all apps, indicating that supporting this feature is a must for any app to be able to compete in this domain. Jodel had the most posts about *anonymity*, but many are actually positive, with users expressing satisfaction at the level of privacy offered by the app. Jodel is one of Yik Yak's alternatives that does not require any sort of sign up to use,

> Love how you don't have to sign up because of anonymity. Sometimes we need to ask questions for people around us and get just the answers. No introduction or getting to know one another.

Firechat also received numerous *anonymity* related reviews, but for the opposite reason. Despite not requiring internet access, Firechat still requires users to create a global, unique profile online before it can be used. This behavior was remarked on angrily by users, with posts such as,

> It's supposed to be decentralized and operate locally and therefore it [should] just store accounts locally instead of signing up through your centralized servers

Entertainment concerns were common in reviews, but extremely rare on Twitter. This may be explained based on the fact that the tweets considered in our analysis were intended to be a direct communication with the app developer, thus, users are less likely to tell the developer "your app is fun". However, despite the lack of tweets, expressions about *entertainment* were still common in reviews, with many posts such as "It's fun!" and "I enjoy using it!". In general, our model shows that a local posting feature would help apps to be more *entertaining* and increase users' *engagement* while using the app.

Posts about *maintaining boundaries* included users discussing the ability to limit their communication with only certain people. In some other cases, users were concerned about the behavior of the app itself. Users particularly wanted apps which were not intrusive, and which did not force them to be aware of activities they would rather ignore. Notifications and advertisements were a common cause of complaint; a Whisper user tweeted *"@Whisper can I pay to remove ads on the app? They're so intrusive and they kill the experience"*. Firechat feared the worst in this regard, with users outraged by push notifications being used as advertisements.

In general, our model suggests that the majority of users are concerned about *toxicity*, at the same time, they do not want to lose their *anonymity*. Therefore, some sort of a

moderation policy is a must for any app trying to compete in this market. However, app developers must carefully consider the impact of that policy on users' freedom of *expression* as strict policies seem to turn users away from the app. Furthermore, our analysis suggests that apps in this domain should be careful about the amount and type of notifications they send to their users. Allowing users to block other users or control what type of notifications to see, or finding less intrusive ways for rolling out ads, would help users to *maintain their boundaries*. Features such as local and away posting are also important to enhance users *entertainment* and *engagement*, while user profiles and mandatory sign-up should probably be avoided as they are perceived by users to be hurting their *anonymity* and overall *usability* of the app.

4.4. Discussion and Expected Impact

The domain model in Fig. 4.3 shows that several in-depth insights can be gleaned from analyzing user concerns along with their relations to the core features of the domain. Specifically, the model provides an effective mechanism for externalizing domain knowledge, providing access to information that can be otherwise invisible to app developers. Through the model's dependency relations, developers can get a sense of the synergy and trade-offs between features and concerns, and thereby adjust their release strategies to focus on features that enhance desirable concerns and mitigate the negative ones. For example, according to our model, a new app in the domain of anonymous social networking apps should start by offering a feature for anonymous posting with some sort of moderation. A second priority feature would be to add local and away posting functionalities to enhance the sense of community (engagement). A low priority feature might include peer-to-peer (P2P) posting. This feature can be postponed to later releases since users seem to be less concerned about it (Currently only supported in Firechat). Similarly, if an existing app wants to attract former Yik Yak users, or to avoid a similar destiny, they can consult our model to get an up-to-date picture of what features to tweak, add, or drop. For instance, existing apps could enforce a more strict moderation policy. However, this policy

should be implemented in such a way that does not overly constrain users' expressiveness. Furthermore, an app which offers anonymous posting should start thinking about dropping the mandatory user registration (profiles) feature to avoid losing users. The model also shows that apps pushing ads on users might need to tweak their monetization strategy to avoid invading users' space (i.e., Maintaining boundaries).

The type of information provided by our model can be particularly useful for smaller businesses and startups trying to break into the app market [123, 43]. Startups are different from traditional companies in the sense that they have to immediately identify and implement a product, often known as the Minimum Viable Product (MVP), that delivers actual customer value [123, 132]. Our model will serve as a core asset that will help startup companies, with little operating history, to get a quick and comprehensive understanding of the history of the domain, providing information about how specific user concerns, features, and their relations have emerged and evolved as apps in the domain have evolved. After release, developers can further use the model to automatically track users and rival apps, reactions to their newly-released features. Such knowledge can then be utilized to make more informed release engineering decisions for future releases of the MVP.

In summary, our case study provides initial evidence regarding the feasibility and value of creating such models for other domains. Our overarching goal is to be able to generate such models automatically for any domain, focusing on active areas of the app market where apps are facing a greater risk of failure or losing their user base to other competing apps. Furthermore, our work has shown the value of leveraging failure in the app market. The ability to analyze individual cases of failure represents a unique opportunity for gaining individual and organizational knowledge about what went wrong, the key learning points, and how to prevent such failures in the future [150].

In terms of threats to validity, the analysis presented in the chapter takes the form of a case study. In empirical software engineering research, case studies are conducted to investigate a single entity or phenomenon in its real-life context within a specific time

space [167]. While case studies can be more realistic and easier to plan than experiments, their results can be difficult to generalize. Furthermore, the majority of the analysis was carried out manually by the authors. Therefore, some of our outcomes might have been impacted by subjectivity. However, case-studies are observational in nature. Therefore, relying on qualitative manual analysis is not uncommon. While these threats are inevitable, we attempted to partially mitigate them by using a systematic process to examine and interpret the multiple sources of information included in our analysis and by having the two coders (authors) working independently to identify the main features of the domain (Table 4.2). We further proposed a systematic feature-concern coding procedure to impose objectivity and enable others to replicate our results.

4.5. Related Work

The research on app store analysis has noticeably advanced in the past few years. Numerous studies have been conducted on app user feedback classification, summarization, and prioritization. A comprehensive survey of these studies is provided in [112]. In this section, we selectively review and discuss important work related to app success and survivability.

Lim et al. [92] introduced *AppEco*, an artificial life simulation which simulates the app ecosystem using simple procedural rules. Apps were represented as sets of features, where each feature can be present or absent. These sets were compared with the desires of virtual users, who downloaded the app if it implemented some of their desired features. The simulation revealed that the most successful strategy by far was simply to copy the features of the most popular apps. The results also showed that more experienced developers optimized their apps better, while copycats did not improve their performance over time.

In [93], Lim et al. followed up on their earlier study in [92] to determine which strategies for presenting apps to users was best for the app ecosystem's health as a whole. In their simulation, users would search through lists of top apps, new apps, and various keyword searches, and would browse apps in order to choose the one to download. The results

showed that the most effective ranking schemes were those that responded the fastest to changing desires. Specifically, ranking searches according the previous day's downloads led to the most relevant results.

Gómez et al. [47] introduced App Store 2.0, a visionary app store which contains a risk analysis tool to locate potential performance and crash problems, aggregate crash logs, and automatically generate reproducible scenarios for bug testing. To test the feasibility of their vision, the authors implemented a prototype app store back-end, released as a set of tools and APIs that developers apply to their apps to generate the necessary raw data for analysis. The risk analysis tool was applied to mine more than $10,000$ potentially crash-prone apps [48]. The crash analysis tool was found to be useful for fixing bugs.

Petsas et al. [133] studied the download distribution of apps to determine the ideal developer strategy for pricing in four popular Android app marketplaces. The analysis revealed that free apps followed a different distribution than paid apps. More specifically, downloads for free apps demonstrated a *clustering effect* with 90% of the downloads going to 10% of the apps. The results also showed that free apps with an ad-based revenue strategy may result in higher financial benefits than paid apps.

In order to help developers stay informed about their app's community, Fu et al. [42] developed Wiscom, a system for discerning major app user concerns and preferences. The proposed system uses Latent Dirichlet Allocation (LDA) [15] to determine the major topics of complaints in negative reviews. The proposed system was evaluated on a high-quality corpus of reviews gathered from over 50,000 apps. The results showed that Wiscom was able to detect the inconsistencies between user comments and ratings, identify the major reasons why users disliked an app, and learn how users' complaints changed over time.

Li et al. [90] analyzed one month of data from the Chinese app store Wandoujia to determine when and why users installed apps. By examining app co-installation rates, the authors found that music apps were often paired with games, and apps for communication often coincided with apps related to video sharing. Uninstallation patterns were also ana-

lyzed to determine the fate of abandoned apps. The results showed that 93% of abandoned apps did not return to prominence (e.g., apps tend to be abandoned by users rapidly and with finality).

Our brief review shows that related work in the literature has focused on either mining user reviews for technical feedback or studying the factors that influenced success in the app market. Our work in this chapter extends existing work along multiple dimensions. First, in our analysis, we focus on extracting and modeling user feedback at a domain, rather than individual app, level. Second, we introduce failure as a main source of knowledge that can be utilized in order to enable a better understanding of the dynamic nature of user requirements in the mobile app ecosystem. Third, we demonstrate the feasibility of integrating human (user feedback on Twitter and the app store), market (download rates and user migration patterns), and system (feature analysis) information to generate a unified domain model that reflects an accurate picture of the current state of the domain.

4.6. Conclusions and Future Work

The case study reported in this chapter systematically analyzed the ripple effect of the failure of a widely successful app on its competitors and their main features. Specifically, in this chapter, we investigated the rise and fall of the social networking app Yik Yak and the impact it left on the app market. Our analysis can be described as a three-step procedure. First, we tracked multiple news outlets, social media platforms, and app download statistics, to identify the main competing apps to which former Yik Yak users migrated. We then performed a qualitative feature-oriented domain analysis, using a direct analysis of the app features and a systematic analysis of user feedback, to identify the main functional features and user concerns of the domain. A domain model was then created, using F-SIG notation, to depict the interrelationships between user concerns and the core features of the domain.

The case study reported in this chapter represents a first-of-its-kind in-depth analysis of modeling user concerns in the app market. Our expectation is that, applying such analysis and modeling at a large scale will provide app developers with a systematic understanding

of their domain as well as help them to predict the impact of feature changes on their end-user acquisition and retention rates. To achieve these long term goals, the work presented in this chapter will be expanded along two dimensions:

- More case studies: Similar studies focused on other cases of success and failure in the app store (e.g., Vine and Pokèmon Go) will be conducted. Our objective is to build a knowledge-base of success and failure in the mobile app ecosystem. Such knowledge will be later used to derive a formal theory that can be used to explain the different factors that control app survival.

- Automated modeling: the qualitative analysis carried out in this chapter was mainly manual. However, repeating such type of analysis over larger domains with hundreds of apps and millions of reviews and tweets can be a time-consuming and laborious task. Therefore, our future work will be focused on automating the process, utilizing existing automatic app store and social media mining methods [54, 72, 105, 112] as well as automated domain modeling techniques [8, 34, 40, 61, 91]. Our goal is to devise automated methods for data collection, domain feature analysis, and model generation. These methods will be evaluated over large datasets of app data to ensure their practicality and accuracy.

Chapter 5. Modeling App Ecosystems

Modern application (app) stores enable developers to classify their apps by choosing from a set of generic categories or genres such as *health*, *games*, and *music*. These categories are static—new categories do not necessarily emerge over time to reflect innovation in the mobile software landscape. With thousands of apps classified under each category, analyzing crowd feedback at such a vast scale can be infeasible and error-prone. To overcome these limitations, in this chapter, we propose an effective technique for automatically analyzing and modeling crowd feedback with more focused categories of functionally-related mobile apps, or micro-ecosystems, in the mobile app store. To realize this goal, we present a case study targeting the domain of food delivery (courier) apps. Specifically, our analysis in this chapter is two-fold. First, we use several classification algorithms to separate and categorize important crowd concerns in the ecosystem, and second, we propose and evaluate an automated technique for modeling these concerns along with their latent interdependencies. Our generated model can help app developers to stay aware of the most pressing issues in their ecosystem, and thus, develop sustainable release engineering strategies that can respond to these issues in an effective and timely manner.

5.1. Introduction

Understanding the specific domain of competition is critical for app survival. Specifically, the clusters of functionally-related apps form distinct *micro-ecosystems* within the app store ecosystem. A software ecosystem can be defined as a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them [69]. Systematically analyzing and synthesizing knowledge at such a micro level is critical for developers to understand the current landscape of competition as well as end-users' expectations, preferences, and needs. Consequently, they must adapt sustainable release engineering strategies that can respond to customers' needs quickly and effectively [33, 41, 62, 152]. However, such knowledge is often tacit, embedded in the complex interplay between the user, system, and market components of the ecosys-

tem. According to organizational knowledge theory [127], to be effectively shared, tacit domain knowledge must be translated into explicit knowledge, a process that is known as *externalization*. Once explicit knowledge is created, it can be preserved, communicated, and passed through to other individuals and organizations [44, 50].

To address these challenges, this chapter proposes an automated crowd-driven approach for micro-ecosystem analysis in the app store. The proposed approach is evaluated through a case study targeting the mirco-ecosystem of food courier, or delivery, apps. These apps, typically classified in popular app stores under the *Food & Drink* category, form a uniquely complex and dynamic micro-ecosystem. This micro-ecosystem (or ecosystem in short) consists of users, drivers, restaurants, and service providers, functioning in an extremely competitive environment and under strict business as well as technological constraints. The main goal of our analysis is to demonstrate how such a complex ecosystem can be automatically analyzed and modeled. Our objective is provide app developers with a set of practical guidelines for assessing the fitness of their mobile apps and understand the main pressing issues in their ecosystems, thus, help them to develop sustainable software evolution strategies that can adapt to market shifts and consumer needs. Specifically, our contributions in this chapter can be described as follows:

- We conduct a qualitative analysis over a large dataset of user feedback, collected from the Twitter feeds and app store reviews of food delivery apps. Our objective is to identify the main user concerns in the ecosystem of these apps and further classify these concerns into more fine-grained categories of technical and business issues.

- We propose and evaluate a fully automated approach for modeling the main business concerns in the ecosystem of food delivery apps along with their latent interdependencies.

The remainder of this chapter is organized as follows. Section 2 provides background information, motivates our research, and discusses our research questions. In Section 3,

we describe our domain scoping procedure as well data collection process. In Section 4, we describe our qualitative analysis and automated classification results. In Section 5, we propose an automated approach for extracting and modeling the relations among the business elements of the ecosystem. In Section 6, we discuss the potential impact of our findings. In Section 7, we describe the main threats to our study's validity. Finally, in Section 8, we conclude the chapter and describe directions of future work.

5.2. Background, Rationale, and Research Questions

After they are published, apps enter a long phase of feature optimization in order to maintain market viability [33, 41, 62, 87, 152, 166]. From an evolutionary point of view, this process is equivalent to acquiring traits that can lower the chances of elimination by natural selection [116]. This *survival-of-the-fittest* effect can be observed in the app store. Specifically, similar to business firms competing in the market, apps are competing actors in an ecosystem of finite resources—only a handful of apps dominate downloads and revenue, leaving only a fraction of market value for other apps to compete over [25, 95, 133, 152]. In such a competitive market, end-user experience and satisfaction play a paramount role in the success of applications. Therefore, mobile app developers, interested in maximizing their revenue, need to find effective mechanisms for monitoring and integrating domain-wide crowd feedback into their release planning.

The research on mining user feedback for app development has noticeably progressed in recent years [112], revealing that user reviews on popular app stores contain substantial amounts of technical information that app developers could benefit from [107, 131, 72, 73, 71]. Social media platforms, such as Twitter, were also found to be an active source of user feedback [165, 57]. In general, such feedback can be classified into three main categories: feature requests (e.g., "could you please add the ability to post pictures"), bug reports ("it keeps crashing when I hit sync"), and miscellaneous (e.g., "I love u", "I hate the new update", etc.).

Despite these advances, the majority of existing research is focused on mining the crowd feedback for individual apps, with little attention paid to how such information can be utilized and integrated to facilitate software analysis at an ecosystem level. Extracting concerns at such a level can be a more challenging problem than focusing on single apps, which typically receive only a limited number of reviews or tweets per day [115]. Furthermore, other types of concerns, often originating from other actors in the ecosystem (i.e., business or service issues) are typically ignored, leaving app developers unaware of important non-technical issues in their ecosystem. These observations emphasize the need for unified crowed-based models that can integrate multiple heterogeneous sources of user, app, and market information to reflect an accurate picture of the current state of the ecosystem.

5.2.1. Motivation and Case Study

To bridge the gap in existing research, in this chapter, we present a case study on modeling the crowd-feedback in micro-ecosystems of functionally-related apps in the app store. Our case study targets the domain of food delivery apps. The first major food courier service to emerge was Seamless, in 1999. A product of the internet boom, Seamless allowed users to order from participating restaurants using an online menu. This service was popular, growing to include caterers as well, while spreading to more cities. Following Seamless, Grubhub was also met with success when it began offering web-based food delivery for the Chicago market in 2004. As smart phones became more popular, a number of new food couriers took advantage of the new demand for a more convenient mobile app-based delivery services. Of these competitors, UberEATS rose to the top, leveraging their experience with the mobile ride-share business model to adapt to food delivery. By the end of 2017, UberEATS became the most downloaded food-related app on the Apple App Store. According to Statista—the online statistics, market research and business intelligence portal—, revenue in the Online Food Delivery segment amounts to 18,358 million US dollars in 2019. This revenue is expected to show an annual growth rate of 7.3%, resulting in a market volume of 24,345 million US dollars by 2023 [6].

The domain of food delivery apps, along with its users (e.g., restaurant patrons), service (e.g., drivers and cars), and business components (e.g., restaurants), represents a uniquely complex and dynamic multi-actor ecosystem. This complexity imposes several challenges on apps operating in this domain. These challenges can be described as follows:

- Fierce competition: users often have multiple services to choose from within a given metropolitan area. Switching from one app to another is trivial, and users are highly impatient with late or incorrect orders.

- Difficult time constraints: food delivery services have less than one hour for delivery. This forces developers to innovate technically to provide faster delivery than their competition.

- Decentralized fulfillment: the drivers are generally independent contractors who choose whom to work for and when to work. This creates challenges, not only for job assignment, but also for predicting when and where human resources will become available.

- Multi-lateral communication: in order to fulfill an order, the delivery app must communicate with users, drivers, and restaurants to ensure that the food order is ready when the driver arrives, and that the user knows when to expect delivery. Each channel of communication presents an opportunity for failure.

These challenges make the set of food courier apps a particularly interesting subject (ecosystem) to be targeted by our case study. The main objective of our study is to demonstrate the feasibility of automated micro-ecosystem analysis in the app store and to provide systematic technical and business feedback for existing app developers as well as newcomers trying to break into the food delivery app market.

5.2.2. Research Questions

To guide our analysis, we formulate the following research questions:

- **RQ_1: What types of concerns are raised by users of food delivery apps?** Mobile app users are highly vocal in sharing suggestions and criticism. Understanding this feedback is critical for evaluating and prioritizing potential changes to software. However, not all concerns, especially in such business-oriented apps, are technical in nature. Therefore, developers must also be aware of business discussions, such as talk of competitors, poor service, or issues with other actors in their ecosystems.

- **RQ_2: Do users raise different issues over different channels of feedback?** In our study, we aim to determine the type, percentage, and value of issues typically raised by users in two main channels of crowd feedback, app store reviews and social media, and investigate how feedback from these different channels contribute to the knowledge-base of ecosystem.

- **RQ_3: Can elements of the software ecosystem be automatically extracted and analyzed?** Assuming that crowd feedback contains useful information, capturing, classifying, and modeling such information at an ecosystem level can be a challenging task. However, this type of analysis will enable us to discover the most important elements of the ecosystem for developers to focus on, as well as their degree of association.

In what follows, we describe our efforts to answer these questions in greater detail.

5.3. Scoping and Data Collection

In order to determine which apps should be considered for our ground truth dataset, we used the *top charts* feature of the Apple App Store and Google Play. Popular app stores use such charts to keep the public aware of the top grossing and downloaded apps in the store. UberEats is the most popular food delivery app on the App Store. Among the top ten apps in the *Food* category, there are three additional competing delivery apps: Doordash, GrubHub, and PostMates. If we broaden our focus to the top twenty-five apps, only one additional food courier app is found: Eat24. Eat24 was recently acquired by

Table 5.1. The number of posts collected for each app from each platform

App	Tweets	App Store (iOS)	Google Play (Android)	Total
Doordash	344	6,685	5,273	12,302
GrubHub	414	1,058	2,863	4,335
Postmates	450	1,467	2,820	4,737
UberEats	625	4,347	18,718	23,690

GrubHub, and have redirected users to their parent app, allowing us to exclude it from the analysis. The Play Store shows the top 25 most popular apps in an arbitrary order rather than specific ranking. However, we find that UberEats and its three main competing apps are also present within the top 25. Therefore, the apps UberEats, Doordash, GrubHub, and PostMates will cover all the most popular food delivery services available on both platforms.

After the list of apps is determined, the second step in our analysis is to identify and classify the main user concerns in the ecosystem. Prior research has revealed that technically relevant feedback can be found in tweets [56, 165] and app store reviews [106, 131, 144]. We used the free third-party service *AppAnnie* [5] to extract reviews. This service allows reviews up to 90 days old to be retrieved from Google Play and the Apple App Store.

Retrieving tweets from Twitter requires the usage of a search query. We searched for tweets directed to the Twitter account of the apps of interest. For example, to retrieve tweets associated with UberEats we searched for `to:ubereats`. Our analysis has shown that this query form yields a large rate (roughly 50%) of meaningful technical feedback among the resulting posts [165]. We scraped tweets directly from the Web page containing the results, going back for 90 days, covering September 4th to December 2nd of 2018. In total, 1,833 tweets, 13,557 App Store reviews, and 29,674 Google Play reviews were extracted. Table 5.1 summarizes our dataset, including the number of tweets, App Store reviews, and Google Play reviews collected for each app in our dataset.

5.4. User Concern Analysis

Under the first phase of our analysis, we are interested in the types of feedback that are available in the collected data. In particular, this phase of analysis can be divided into two main steps. First, we conduct a qualitative analysis to determine the presence and distribution of different types of user concerns in the different sources of crowd feedback and over the different apps in our ecosystem. Second, we examine the performance of multiple classification algorithms for automatically classifying our data into different categories of crowd concerns.

5.4.1. Qualitative Analysis

To conduct our qualitative analysis, we sampled 900 posts (300 tweets, 300 iOS reviews, and 300 Android reviews) from the data collected for each app in our domain. To perform the sampling, we developed a Ruby program to first execute a `shuffle()` method on the lists of tweets and reviews to randomize the order. The first 300 posts from each source of user feedback were then chosen. To analyze the sampled data, we manually went through the set of tweets and reviews for each app, identifying the main concerns raised in these posts as they appeared in the text. The categorization of the data was then manually examined by an external judge for validation. In general, the following categories and sub categories of concerns were identified in the user feedback sampled from our dataset of food delivery apps:

- **Business concerns:** this category includes any concerns that are related directly to the business aspects of food delivery. In general, these concerns are subdivided into two main subcategories:
  - **Human:** these concerns are related to interactions with employees of the courier apps. Users often complained about orders running late, cancellations, and drivers getting lost on the way to delivery. Statements such as:

> Placed order, restaurant had order ready quickly, no driver ever showed up

were common. Service-related reviews were on average the longest, often narrating multi-paragraph sequences of human (mainly driver) failures that lead to undesirable outcomes. This category also included posts praising the service provided by the app, such as:

> Convenient! Overall orders have been good and speedy. Restaurants usually beat the estimated delivery time by a bit, which is nice.

– **Market:** the courier apps in our dataset generally make money either through flat-rate delivery charges or surcharges added to the price of individual menu items. Users are highly sensitive to the differences between what they would pay at the restaurant versus at their doorstep. Posts such as:

> Besides the regular $4.99 booking fee they now charge a busy area fee, in my personal experience was $13.47. Crazy to have to pay so much when GrubHub or Seamless or other apps offer free delivery.

were relatively common, while posts complimenting low fees and markups were rare. Price complaints were not the only form of market-related feedback. Other posts included generic discussions of market-related concerns such as business policy (such as refunds), discussion of competitors, promotions, and posts about participating restaurants and delivery zones. Requests for service in remote areas were fairly common, such as:

> I'm sure it would be a good app if it worked in my area, which it doesn't or in the next town either.

- **Technical concerns:** this set of concerns includes any technical issues that are related to the user experience when using the app itself. In general, technical concerns can be classified into three main subcategories:

  - **Bug reports:** posts classified under this category contain descriptions of software errors, or differences between a user's expectations and observed app behavior. Bug reports commonly consist of a simple narration of an app failure. In our dataset, we observe that the most common bugs relate to menu correctness, application of payment credits, and app crashes. For example:

    > Grub hub took over Order Up, which I used to get away from Grub Hubs horrible selection of restaurants in my area. And now I can't even order food because their app keeps crashing.

  - **Feature requests:** these posts contain requests for specific functionality to be added to the app, or discussion of success/failure of distinct features. For example, some users of DoorDash complained about being forced to tip before the order was delivered. Users of Eat24 lament a recent update which removed the ability to reorder the last meal requested through the app, such as in a review stating:

    > *"The old app let me reorder old orders with one click. The new app can't do that at all. Explain how that's better".*

    Under this category, we also include non-functional requirements (NFRs), or aspects of software which are related to overall utility of the app (e.g., usability, reliability, security, and accessibility), rather than its functional behavior [45, 29, 111]. Ease-of-use was the most common NFR cited by users, followed by user experience (UX). For example:

Table 5.2. The number of posts (tweets and reviews) classified under each category of user feedback

	Tweets	**iOS**	**And.**	**All**
Human	443	649	276	1368
Market	392	563	340	1295
Business	704	931	522	2157
Bug	244	175	114	533
Feature	54	106	77	237
Technical	292	258	186	736

> They changed the app. Now the UX is garbage. Cannot filter by delivery fee, so now I have to click into each restaurant to view the delivery fee.

- **Miscellaneous:** these posts were considered not relevant to developers in our ecosystem. Specifically, this category included spam, generic news items, and context-free praise and criticism, such as *"I hate this app!"* and *"This app is great!"*. A handful of posts which were marked as *Miscellaneous* included drivers themselves discussing their social experiences working for the app. While these posts might provide potentially valuable information to the developers, few contained actionable technical or business feedback.

5.4.2. Discussion

Table 5.2 shows the number of posts classified under each category of concern in the sampled dataset. It is important to point out that our identified categories were considered orthogonal: each post could be any combination of human, market, bug, and feature issues. Therefore, there was considerable overlap between categories. Fig. 5.1 shows the percentage of overlap between the different categories.

Furthermore, our analysis in Fig. 5.2 shows that, based on the total number of relevant posts, Android reviews were the least informative in comparison to other sources of feedback. One potential explanation for this phenomenon is that Google Play does not

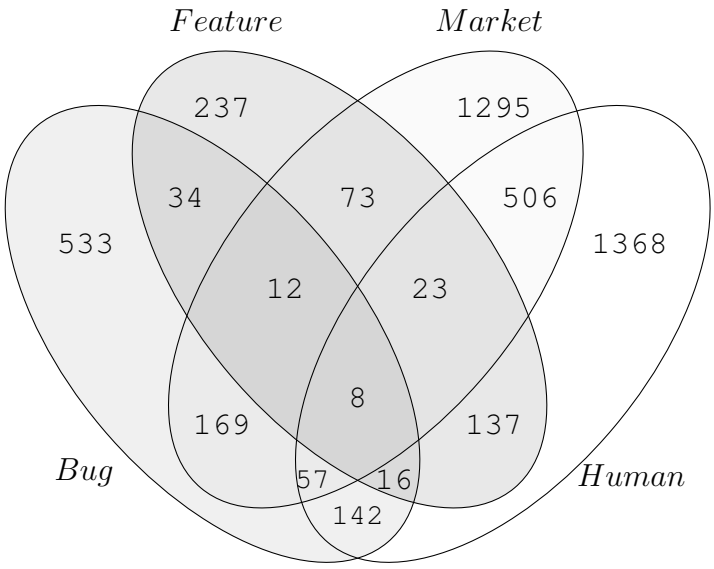

Figure 5.1. A Venn diagram of the distribution of classification labels and their overlap in the dataset.

pose any restriction on the number of times an app can request users to leave a review for the app, while the Apple App Store limits app in this respect. As a result, many Android reviews were terse, with statements such as *"I'm only posting this because the app keeps nagging me"* being common. The results also show that the distribution of concerns over the apps was almost the same. As Fig. 5.3 shows, concern types spread almost equally among apps, highlighting the similarity between the apps in their feature and user base.

In terms of the nature of specific ecosystem-wide crowd concerns, we make the following observations about the data:

- There is substantially more overlap between business-relevant posts than technical posts. One reason for this is that complaints about customer service and drivers (human issues) were very common, and such posts would often come with an exhortation to use a competing service or further complaints about refund policy (market-related). For example:

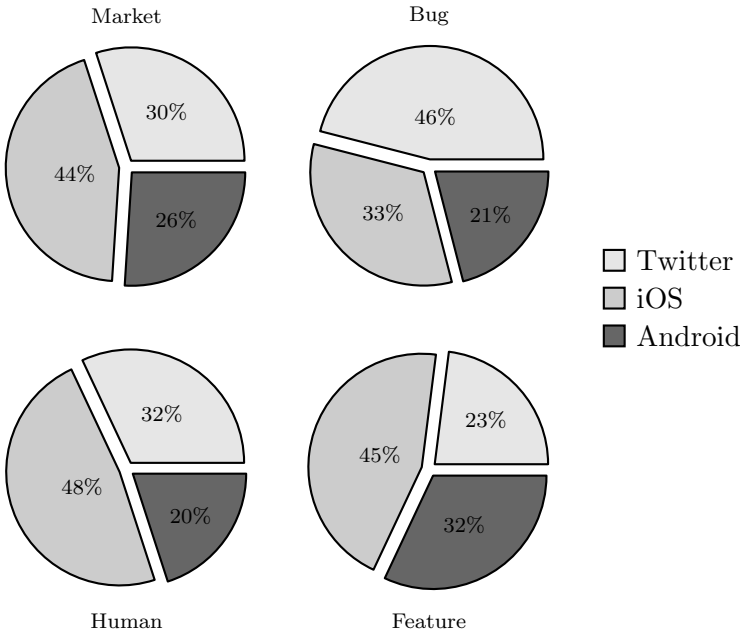

Figure 5.2. The distribution of concern category over each source of user feedback.

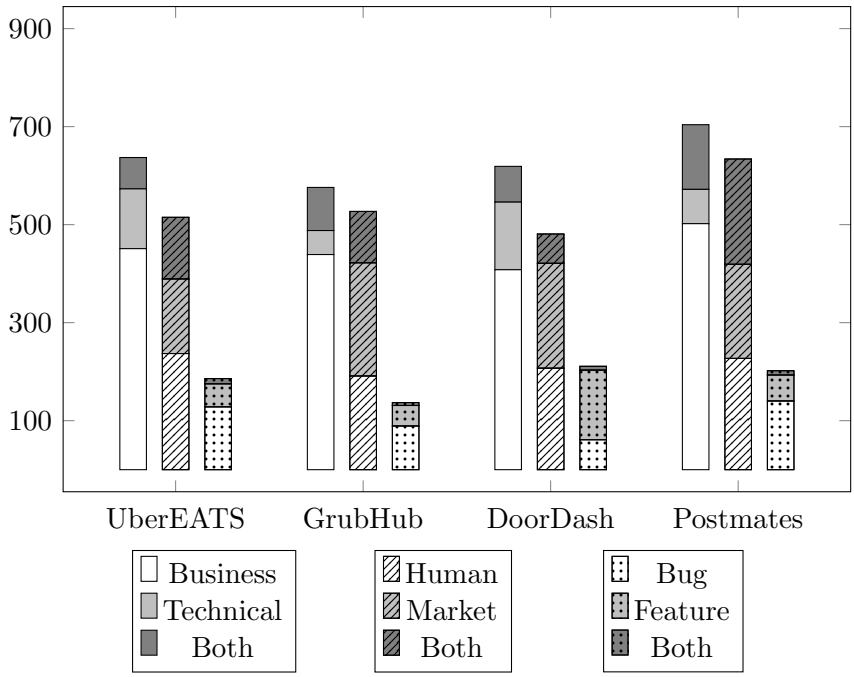

Figure 5.3. The distribution of concern categories and subcategories for each app.

> I'm going to stop ordering through you guys [DoorDash]. My order Friday was missing items. Instead all you guys did was offer a $7 refund.

- Similarly, positive posts often remarked on how both restaurant selection and friendliness of drivers were impressive:

> Easy to use!. Great options for delivery in my area and fast friendly service. The app is easy to use and offers great choices.

- The single most common problem was with drivers. Specifically, drivers would be dispatched inefficiently, causing long wait times. Users were especially upset when their app showed drivers going in the wrong direction. For example:

> In addition, the address that I gave to UberEats took the driver to a completely different parking lot...

- Another common class of complaint was that of problems with customer service. Users often expressed dissatisfaction with how difficult it was to find service numbers, and how long it took to receive answers:

> Do not ever use this service! The contact number is nowhere to be found; I had to ask Google to find it.

- Users were often frustrated to discover that services would generally only offer refunds for the delivery charge, rather than for food, even if the food was rendered inedible by an unreasonably long delivery time.

> They were "unable to get me a refund" for food that arrived cold and rubbery when I live 3 minutes away from the restaurant. They think that throwing a 10$ giftcard at you will make you be quiet after spending over 20$ on inedible food.

- Technically-relevant posts, on the other hand, had less overlap. There simply were not many instances of requesting new functionality while complaining of existing mistakes. A common bug report was promotion codes not being applied to orders correctly. For example,

> The promo code was rejected, inaccurately saying that I was not eligible because a previous order had been made.

- Another frequent report was service outages. Twitter was the most common platform to report app connection issues. Tweets such as *"The servers are down!"* and *"Great timing for an outage"* occurred often.

- Bugs would sometimes stem from failed communication between the delivery service and the restaurant, especially regarding menu items and hours-of-operation. This would result in tweets such as:

> Postmates so I ordered baby blues spent 52$ for my postmate to send me a picture of the place closed so I had to cancel my order and now I cant get food tonight

- Security errors were surprisingly common. Users would often find unexplained charges to their account:

> Postmates my account was hacked. I reset my password and people all over the country are still ordering on my account/credit card.

- Occasionally the GPS systems in the drivers' apps would completely fail, causing the drivers to ask the user for help. Many users were upset when this happened:

> Driver got lost had to ask me for BASIC directions, then drove in the complete opposite direction. The food came so late it was inedible.

- Users often complained about poor communication regarding order delays. Sometimes, services failed to route a driver to an order, and rather than alert the customer, they gradually pushed the delivery window back. Users would often end up worse off than if they had never used the service to begin with, as the restaurant could close in the meantime.

  > I had to contact grub hub, not the other way around, about a delivery that was an hour beyond the delivery window and the estimated time kept pushing further back.

- Some services, such as Doordash, allow drivers to combine deliveries to improve throughput. This decision angers consumers, however, with posts such as the following being common:

  > @doordash no one wants to wait an hour for food just because a dasher has the option to make a detour and deliver 5 other things first.

5.4.3. Automated Classification

Under this phase of our analysis, we examine if the different categories of user concerns can be automatically separated. Specifically, we examine the ability of three different classification algorithms, commonly used in text classification tasks, to automatically detect the different types of issues raised by users in the domain of food delivery apps. Our classification settings can be described as follows:

- **Classification algorithms:** To classify our data, we use Support Vector Machines (SVM), Naive Bayes (NB), and Random Forest. A description of SVM and NB can be found in Section 2.4.1. Random Forest (RF) is a stochastic machine learning algorithm that makes use of a committee of decision trees to classify data instances [63]. Each random tree in the forest is constructed using a conventional decision tree

algorithm (e.g., ID3) along with a combination of *bagging* and random attribute selection [22]. Bagging is the process of choosing a subset of data instances randomly to form a training set. Random attribute selection allows the generated trees to exploit attributes other than those that generate the maximum information gain. Classification decisions are then made according to the consensus of the trees in the forest.

To train our classifiers, we use 10-fold cross validation. This method creates 10 partitions of the dataset such that each partition has 90% of the instances as a training set and 10% as an evaluation set. The benefit of this technique is that it uses all the data for building the model, and the results often exhibit significantly less variance than those of simpler techniques such as the holdout method (e.g., 70% training set, 30% testing set).

- **Text pre-processing:** English stop-words were removed and stemming was applied to reduce words to their morphological roots. We use Weka's built-in stemmer (`IteratedLovinsStemmer` [103]) and stop-word list to pre-process the posts in our dataset.

- **Sentiment Analysis:** we further introduce sentiment analysis as a classification feature. An assumption is that, different categories of user concerns are expressed using different sentiments [164]. Sentiment analysis determines whether a text conveys positive, neutral, or negative feelings. To conduct our analysis, we used SentiStrength (Sec. 2.4.3) [154]. To convert SentiStrength's numeric scores into these categories, we adapted the approach proposed by Jongeling et al. [75] and Thelwall et al. [153]. Specifically, a post is considered positive if $p + n > 0$ where p is the positive score and n is the negative score. $p + n < 0$ indicates a negative score and $p + n = 0$ is neutral. The proportion of sentiments assigned by SentiStrength are displayed in Fig. 5.4. In general, the results show that miscellaneous posts (posts not business or

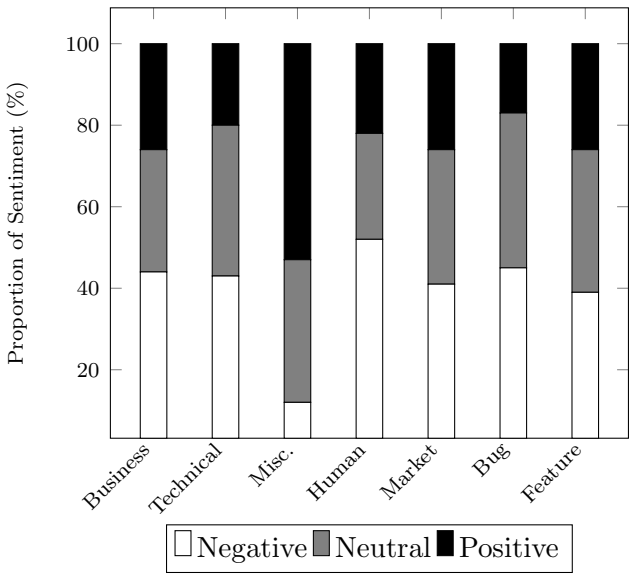

Figure 5.4. The distribution of sentiment over the different types of posts.

technically-relevant) were detected as having more positive sentiment than any other category. This result is expected; non-miscellaneous posts normally describe problems the user is having. Otherwise, the categories had substantially similar sentiment scores overall.

- **Text representation:** To classify our data, we experimented with simple bag-of-words with lowercase tokens. The Bag-of-words representation encodes each post as a vector. Each attribute of the vector corresponds to one word in the vocabulary of the dataset. A word was included in the vocabulary if it were present in at least two posts. Words that appear in a single post are highly unlikely to carry any predictive value to the classifier. An attribute of one in a post's vector indicates that the corresponding word is present, while a zero indicates absence. This representation can be extended to treat common sequences of adjacent words, called N-Grams, as a single word; N is the number of adjacent words, so two adjacent words are a 2-gram. For example, the phrase "this app is good" contains four words, and three 2-grams ("this app", "app is", "is good"). Fig. 5.5 illustrates how bag-of-words and N-gram representations work in practice. "Updated", "app", and "crashes" are words in the

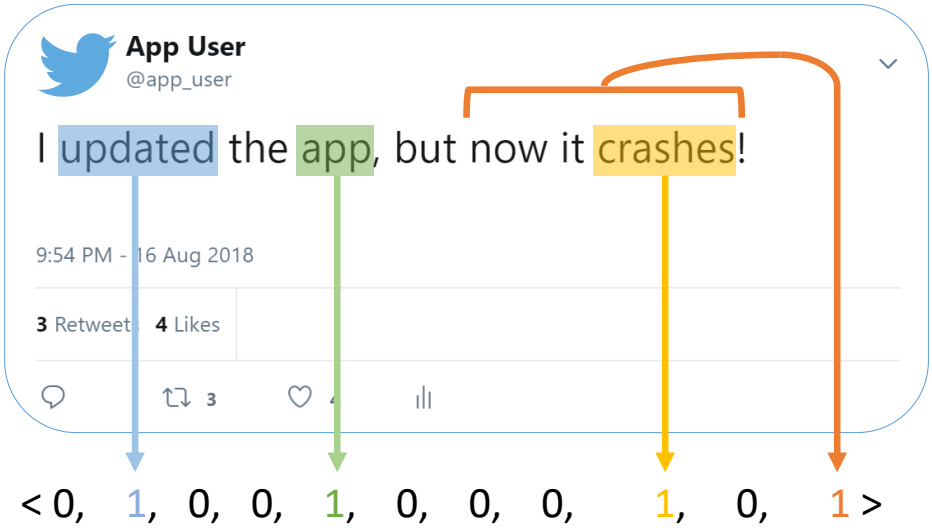

Figure 5.5. A visual representation of an N-Gram encoded tweet.

dataset vocabulary that occur in the tweet. "Now it crashes" is a 3-gram that is also included. Each '1' in the vector representation at the bottom corresponds to one of the highlighted N-grams, while each '0' corresponds to a vocabulary word that was not found in the tweet. To generate this representation, we utilized the N-gram tokenizer in Weka, which allowed 1-gram, 2-gram, and 3-gram tokens to be included in a single dataset.

5.4.4. Classification Results

Table 5.3 shows the performance of NB, SVM, and RF in terms of F_2. In general, stemming and stop-word removal made a small impact, and SVM provided the best overall classification performance, achieving an F_2 of 0.65 in separating the different types of concerns, in comparison to an F_2 of 0.56 and 0.4 for NB and RF respectively. RF was evaluated with 100 iterations. Raising iterations above this number did not improve the performance.

In general, business-relevant posts were easier to classify than technically-relevant posts. This phenomenon is driven by the quantity of each class. Table 5.2 shows that technical posts were rare. The prior-probability of any given post being technical is less than 25%, negatively impacting the performance of all three classifiers. This problem was exacerbated

Table 5.3. A comparison of the performance of the classifiers (SVM, NB, and RF) with lower-casing (LC), stemming (ST), stop-word (SW) removal, and sentiment analysis (SEN).

	NB			**SVM**			**RF**		
	P	R	F_2	P	R	F_2	P	R	F_2
Business									
LC	.85	.79	.82	.89	.85	.87	.85	.87	.86
LC + SEN	.85	.79	.82	.89	.85	.87	.86	.87	.86
LC + SW	.83	.84	.83	.89	.85	.87	.88	.86	.87
LC + SW + ST	.84	.83	.84	.89	.85	.87	.87	.88	.87
Human									
LC	.68	.79	.73	.83	.79	.81	.85	.70	.77
LC + SEN	.68	.78	.73	.83	.79	.81	.85	.69	.76
LC + SW	.69	.83	.75	.83	.79	.81	.87	.74	.80
LC + SW + ST	.68	.81	.74	.83	.79	.81	.86	.75	.80
Market									
LC	.56	.69	.62	.72	.66	.69	.78	.47	.59
LC + SEN	.56	.69	.62	.73	.67	.70	.80	.43	.56
LC + SW	.55	.75	.63	.75	.67	.71	.81	.54	.65
LC + SW + ST	.56	.74	.64	.75	.68	.71	.79	.53	.64
Technical									
LC	.38	.67	.49	.60	.55	.57	.93	.07	.13
LC + SEN	.38	.67	.49	.59	.55	.57	.95	.05	.10
LC + SW	.42	.69	.52	.58	.52	.55	.88	.22	.35
LC + SW + ST	.39	.71	.51	.61	.55	.58	.91	.16	.27
Bugs									
LC	.31	.65	.42	.57	.54	.56	1.0	.03	.05
LC + SEN	.31	.65	.42	.56	.53	.55	1.0	.03	.06
LC + SW	.34	.66	.45	.56	.50	.53	.91	.14	.24
LC + SW + ST	.33	.68	.45	.53	.48	.51	.98	.11	.19
Features									
LC	.17	.65	.27	.41	.38	.40	.00	.00	.00
LC + SEN	.17	.63	.27	.42	.38	.40	.00	.00	.00
LC + SW	.19	.62	.29	.42	.33	.37	1.0	.01	.03
LC + SW + ST	.18	.67	.28	.38	.30	.34	1.0	.01	.02

for the individual technical categories, with features only occurring in 6.5% of posts. One of the reasons that technical posts were rare compared to other domains, is that users had so many more business-related issues to discuss. Food courier services would often fail behind the scenes, causing drivers to be dispatched to incorrect locations, or customer support to fail to call. These failures often cause customers to discuss competition and pricing. As a result, business concerns *crowded out* technical concerns. In other domains, failures are more immediate and consumer visible, meaning that user concerns are more likely to take the form of bug reports.

In our analysis, we further tested the bag-of-words representation, and then allowing 3- and 3-grams to be included alongside individual words. Neither approach improved the performance. Table 5.4 shows a comparison between a 1-gram encoding (i.e., bag-of-words), and an encoding which included 2- and 3-grams. The lack of improvement partly stems from the fact that the additional tokens often had the same class implications as their constituent words. For example, the term *account* was found to have a negative implication on the *business* class—meaning, that posts containing the word *account* were unlikely to be business-related. Most of the related N-grams, including *account got hacked* and *account was hacked* had the same implication, except with a substantially smaller weight. Therefore, they were essentially irrelevant to classification. In some other cases, 2- and 3-grams did not have the same implication as their constituent words. For example, *promo* was positively implicated to *business*, but *promo code* had a negative implication. However, the single word in this case, and in many others, had a higher weight than the bi- and 3-grams, and occurred in substantially more posts. Often times, the 2-grams had the same weight and occurrence as the 3-grams, making the 3-grams superfluous.

Finally, our results also show that the sentiment polarity of posts had almost no impact on the classification accuracy. This can be explained base on the fact that the different categories and sub-categories of crowed feedback had substantially similar sentiment scores overall.

Table 5.4. A performance comparison of SVM using ordinary bag-of-words vs. N-grams.

	1-, 2-, and 3-Grams			**Bag-of-words**		
	Prec.	**Recall**	F_2	**Prec.**	**Recall**	F_2
Business	.89	.85	.87	.89	.85	.87
Human	.84	.80	.82	.83	.79	.81
Market	.71	.66	.69	.72	.66	.69
Technical	.61	.55	.58	.60	.55	.57
Bug	.58	.52	.55	.57	.54	.56
Feature	.48	.32	.38	.41	.38	.40

5.5. Modeling the Ecosystem

In this section, we propose a procedure to model the business aspects of our ecosystem. Specifically, our analysis can be divided into two main phases: data pre-processing and model generation.

5.5.1. Pre-processing

Under this phase, we perform standard text pre-processing analysis to extract important information and reduce noise in our textual data (business-related reviews and tweets). To conduct such analysis, we utilize the Natural Language Toolkit (NTLK), a Python-based suite of libraries and programs commonly used for symbolic and statistical natural language processing [14]. NLTK provides interfaces to perform Part-of-Speech (POS) tagging, stemming, and stop-word removal, which are important facets of our modeling technique. Specifically, our text pre-processing steps can be described as follows:

- **Data representation:** we limit our analysis to posts that were identified by our classifier as business-relevant. These posts are stored in ARFF format, a common text-based file format often used for representing machine learning datasets.

- **Tokenization:** After individual posts are loaded into NTLK, individual posts are tokenized. Tokenization in our analysis is carried out using regular expressions. Regular expressions can be used to identify individual words in the dataset, ignoring any

special characters (e.g., # and in tweets) that might hinder our POS analysis. Code Listing 5.1 describes the Python script we used to tokenize our posts.

```
#tokenize on most punctuation, with the exception of "'"
tokenizer = nltk.tokenize.RegexpTokenizer(
    \'[\w\\\']+|[\.\,\;\:\"\(\)\[\]\{\}\?\!]+|\S+')

def tokenize( str ): return tokenizer.tokenize( str )
```

Code 5.1. The Python script used in tokenizing our posts.

- ***Part-of-Speech* (POS) Tagging:** We utilize NLTK's part of speech (POS) tagging library to attach POS labels to each word in the dataset. This library uses *skip grams*, or the sequence of words around the word of interest, to determine the POS of a given token. For example, a word will be likely considered a noun if it is preceded by *the*. The Python script in Listing 5.2 is used to conduct our POS analysis.

```
tagged_sentences = [ nltk.pos_tag(sent) for sent in sentences ]
```

Code 5.2. The Python script used in our POS analysis

- **Stop-words Removal**: We apply stop-word removal to avoid including words in our model that do not carry any meaningful semantic content about our ecosystem. English stop-words are contained in a list distributed with NLTK. We augment this list with the names of the apps in our dataset. In addition, we added *quoteless* versions of each word as people in online discussions commonly spell words such as *"you've"* as *"youve"*. Stop-words are only deleted after the POS analysis because these words are important for determining the function of semantically meaningful words. The Python script for removing stop-words is shown in Code Listing 5.3.

- **Lemmatization:** we apply *lemmatization* to reduce the morphological variants of words in our dataset down to their base forms. For example, *drink*, *drinks*, *drinking*, *drank*, and *drunk*, are all transformed to simply *drink*. By applying lemmatization,

```
stopwords = nltk.corpus.stopwords.words('english') + \
["i've", "they've", "they're", "ubereats", "grubhub",
   "doordash", "postmates", "uber", "grub", "post", "dash"]

#add a version of each word without single quote.
stopwords += [ \
  re.sub( "'", '', word ) for word in stopwords ]
```

Code 5.3. The Python script used for removing English stop-words.

we avoid the problem of morphological variants being treated as entirely different words by our model. After lemmatization, we merge words together under each part of speech category. For example *drive* and *drives* are merged to simply *drive* when used as verbs. However, the word *drive* can also be a noun (e.g., "that was a long drive"). Therefore, we only merge words within the same part of speech to avoid losing this semantic distinction. The Python script in Code Listing 5.4 describes how this process is carried out in NLTK.

```
#Helper function to lemmatize a single (word, POS) pair
def lemmatize_one( pair, part_of_speech ):
  return ( lem.lemmatize( pair[0], \
    pos=part_of_speech ), pair[1] )
```

Code 5.4. The Python script used for lemmatization.

Example: the following example shows the impact of applying of our different text pre-processing steps over the user tweet "You've got a great selection of restaurants. #hunger".

Original text: You've got a great selection of restaurants. #hunger

Tokenization: Youve, got, a, great, selection, of, restaurants, #hunger

Removing special characters: Youve, got, a, great, selection, of, restaurants, hunger

Removing stop-words: ~~Youve~~, got, ~~a~~, great, selection, ~~of~~, restaurants, hunger

POS tagging: (got, VBD) (great, JJ) (selection, NN) (restaurants, NNS) (hunger, NN)

lemmatization: (get, VBD) (great, JJ) (selection, NN) (restaurant, NNS) (hunger, NN)

5.5.2. Model Generation

The main task under this step of our analysis is to generate a model the captures the most important entities of our ecosystem along with their relations. Models provide a framework for explicitly describing abstract salient concepts in a specific domain and formally reasoning about these concepts in order to create new knowledge [44, 65, 104]. Formally, a model can be described as an undirected graph $G = \langle \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$, where $\mathcal{C}$ is the set of ecosystem entities, $\mathcal{F}$ is the set of functions associated with these entities, and $\mathcal{R}$ is the set of relations connecting these entities. Technically, we follow the following steps to generate our model:

- **Identifying model entities:** In order to specify the main entities (nodes) of our model, we look for *important* words in our set of reviews and tweets collected for the ecosystem. Our assumption is that such words capture the essence of user concerns in the ecosystem. In Object Oriented software design, when generating conceptual models from requirements text or any textual data, nouns are considered candidate classes (objects), verbs are considered either candidate operations (functions), while adjectives commonly represent attributes [7, 39]. Based on these assumptions, we only consider important nouns, verbs, and adjectives in our analysis. Specifically, we rank these parts of speech based on their Hybrid TF.IDF scores. Introduced by Inouye and Kalita [68], the hybrid TF.IDF approach relies on the frequency of a word to determine its importance to the collection. Formally, the TF part stands for *term frequency*. It is computed as its frequency in the entire collection of posts ($f(w_i)$) divided by the number of words in all posts (N), such that $TF(w_i) = \frac{f(w_i)}{\lg N}$. This modification over classical single-document TF is necessary to counter the short nature of tweets and reviews [68]. The IDF part is used to account for the scarcity of words across all posts by using the inverse document frequency (IDF) of words. IDF penalizes words that are too frequent in the text. Formally, TF.IDF can be computed

as:

$$TF.IDF = TF(w_i) \times \lg \frac{|R|}{r_j : w_i \in r_j \wedge r_j \in R} \tag{5.1}$$

where $TF(w_i)$ is the term frequency of the word w_i in the entire collection, $|R|$ is the total number of posts in the collection, and $r_j : w_i \in r_j \wedge r_j \in R$ is the number of posts in R that contain the word w_i. The purpose of TF.IDF is to score the overall importance of a word to a particular document or dataset. If a word appears very rarely in general (low IDF), but very often in a small number of documents, then such a word will receive a very high TF.IDF score. Words with high scores on average are important to the dataset as a whole. After defining TF.IDF, we have the ability to extract important nouns and verbs from the dataset as a whole. The top 10 nouns, verbs, and adjectives in our dataset as ranked by TF.IDF score are shown in Table 5.5.

- **Identifying model relations:** To generate our model relations, our model generation algorithm considers the co-occurrence statistics of words in the data. For example, in our dataset, the words *customer* and *refund* appear in a very large number of user reviews and tweets. Therefore, the model should have a relation to connect these two entities. To count for such information, we use Pointwise Mutual Information (PMI).

  PMI is an information-theoretic measure of information overlap, or statistical dependence, between two words [28]. PMI was introduced by Church and Hanks [28], and later used by Turney [156] to identify synonym pairs using Web search results. Formally, PMI between two words w_1 and w_2 can be measured as the probability of them occurring in the same text versus their probabilities of occurring separately. Assuming the collection contains N documents, PMI can be calculated as:

$$PMI = \log_2\left(\frac{\frac{C(w_1,w_2)}{N}}{\frac{C(w_1)}{N}\frac{C(w_2)}{N}}\right) = \log_2\left(\frac{P(w_1,w_2)}{P(w_1)P(w_2)}\right) \tag{5.2}$$

Table 5.5. The top 10 nouns, verbs, and adjectives in our dataset ranked by TF.IDF.

Nouns	**Verbs**	**Adjectives**
order	get	first
food	order	good
service	use	great
app	say	terrible
delivery	deliver	horrible
time	charge	wrong
customer	wait	last
driver	cancel	bad
restaurant	give	free
money	want	easy

where $C(w_1, w_2)$ is the number of documents in the collection containing both w_1 and w_2, and $C(w_1)$, $C(w_2)$ are the numbers of documents containing w_1 and w_2 respectively. Mutual information compares the probability of observing w_1 and w_2 together against the probabilities of observing w_1 and w_2 independently. Formally, mutual information is a measure of how much the actual probability of a co-occurrence of an event $P(w_1, w_2)$ differ from the expectation based on the assumption of independence of $P(w_1)$ and $P(w_2)$ [19]. If the words w_1 and w_2 are frequently associated, the probability of observing w_1 and w_2 together will be much larger than the chance of observing them independently. This results in a PMI > 1. On the other hand, if there is absolutely no relation between w_1 and w_2, then the probability of observing w_1 and w_2 together will be much less than the probability of observing them independently (i.e., PMI < 1).

PMI is symmetrical; the amount of information acquired about w_2 from observing w_1 is equivalent to the amount of information acquired about w_1 when observing w_2. The value of PMI can go from $-\infty$ to $+\infty$, where $-\infty$ indicates that the two words are not related, or do not appear together in any of the system's artifacts, and $+\infty$

Figure 5.6. The key elements of the entity-action-property relations represented by our model.

indicates a complete co-occurrence between the words. PMI is intuitive, scalable, and computationally efficient [122, 117]. These attributes have made PMI an appealing similarity method to be used to process massive corpora of textual data in tasks such as short-text retrieval [117], Semantic Web [156, 145], source code retrieval [81, 110], and general text and data mining [66].

To generate the relations in our model, we computed the PMI between every pair of words to determine their relatedness. It is important to point out that, one potential pitfall of relying on PMI as a measure of relatedness, is that the highest PMIs will be found when one word only occurs a single time. This happens often with misspellings and irrelevant words. In order to prevent this phenomenon, we restrict our analysis to only words that occur at least ten times. Ten was chosen due to being the point at which sensitivity to additional increases became less noticeable (i.e., changing 10 to 11 would not substantially alter the results).

- **Model generation:** to generate our model, we extract the top 10 nouns ranked by TF.IDF. We then use PMI to extract the three most associated verbs and adjectives with each noun. An example of a node, or an atomic entity in our model, is shown in Fig. 5.6. This node consists of three main parts:

  1. Entity: the middle part of the node represents the entity's name (*food*), which is basically one of the important nouns (based on TF.IDF) in our dataset.

2. Properties: directly attached to the entity's name from the right is the top three adjectives associated with the entity (based on PMI). In our example, *food* could be *cold*, *hot*, or *ready*.

3. Action: on the left side of the node, we attach the list the top three verbs frequently associated (based on PMI) with the noun (entity's name). In our example, the actions *arrive*, *deliver*, and *come* are commonly associated with the word *food*.

The outcome of our model generation algorithm is shown in Fig. 5.7.

5.5.3. Model Evaluation and Interpretation

To evaluate our model, we assess its ability in informatively expressing the main business-relevant concerns in our ecosystem. Specifically, in what follows, we measure the extent the noun-verb-adjective associations in our model capture the main concerns identified in our qualitative analysis (Sec. 5.4.2.).

- **Drivers**: Drivers were a critical component of the ecosystem. All services struggled with driver *timing*, *directions*, and *friendliness*. Users complained about drivers *combining orders*. The model successfully identified <*driver, find*>, due to the presence of users discussing a driver's inability to find their house (therefore also <*driver, directions*>). The model's representation of the time estimates being inaccurate for drivers is captured in <*delivery, estimate*>. Lack of driver friendliness is captured in <*driver, awful*>.

- **Restaurants**: Restaurants were identified, both in terms of *selection* and *communication*. Users often asked services to add new restaurants, as well as discussed problems that occurred between the app, restaurant, and driver. The relations <*restaurant, show*> is expressed in the model partly due to users stating that the restaurant they want does not "show up" in the app. However, this phrase is more

often associated with the *driver* not appearing at the restaurant. Communication is expressed with the <*restaurant, call*> association.

- **Customer Service**: Customer service was identified as an important concern when an *order* was not delivered on *time*, or *accurately*. Customers also complained about service in the context of not offering *refunds*, as well as giving general negative judgments of the overall customer service experience. In addition, customers sometimes struggled to *find* customer service numbers to begin with. The model identified both *customer* and *service* as important nouns and <*customer, refund*> was successfully identified along with <*customer, incorrect*> for accuracy. Both *customer* and *service* were associated with adjectives *poor* and *terrible* in the model. However, the ability to find numbers to call was not reflected.

- **Orders**: Orders were associated with *delays*. Additionally, users complained about receiving *cold food* as a result. Users were angered by food *waiting* at the restaurant to be picked up. The model identified <*order, leave*> in the context of dispatch and pickup times for orders. In addition, <*order, cold*> and <*order, hot*> were identified successfully.

- **Delivery**: Delivery was associated with a number of complaints about "slipping estimates" and incorrect "estimated times", explaining the relation <*delivery, estimate*> and <*delivery, estimated*>. The relation <*delivery, prepare*> occurred due to issues with orders being stuck in the preparation stage and never being dispatched from the restaurant for delivery. The relation <*delivery, choose*> primarily occurred in the context of users stating that they would "choose a different delivery service".

- **Time**: Time was primarily found in complaints about delivery delays. The relations <*time, estimate*> and <*time, prepare*> appeared for the same reasons as the ones associated with *delivery*. A common occurrence was <*time, waste*> due to unexpected

delays and order cancellations. The pair <*time, long*> occurred in similar contexts, as in "it took longer than the estimated time".

- **Food**: Food was directly related to arrival as captured in the relations <*food, arrive*>, <*food, deliver*>, and <*food, come*>. Food was also associated with temperature, mainly due to the number of complaints about receiving cold food. Complaints that orders had been sitting at the restaurant without being picked up by a driver were common, explaining the association relation <*food, ready*>.

- **App**: App appeared alongside comments about ease-of-use, resulting in <*app, easy*>. The relation <*app, ridiculous*> was a general complaint about poor policies or usability. The relation <*app, delete*> appeared when users discussed deleting an app after a poor experience. A common association was <*app, look*>, appearing due to phrases such as "look into this" and "looks like". The phrase "ends up" resulted in the relation <*app, end*>, particularly when users "ended up" eating cold or incorrect food, or not eating at all.

- **Money**: Money was captured in the relation <*money, waste*> mainly due to users ordering food that ended up being inedible and being unable to obtain a refund, also yielding the model relation <*money, refund*>. Additionally, the verb *take* was associated with phrases like "you take my money but did not deliver" in the pair <*money, take*>.

Table 5.6 shows a mapping between the business entities and their relations that we manually identified in our qualitative analysis versus the ones captured in the model. In general, our evaluation shows that the majority of business concerns identified through our qualitative analysis were actually captured in the model. In fact, several entities that the model revealed were not obvious during our qualitative analysis, thus, giving our model an advantage of being more comprehensive. It is important to point out that more domain entities and their relations can be identified by simply changing the TF.IDF and PMI

thresholds used to generate our model. For instance, due to space limitations, we only considered the top 10 entities in our model. Table shows the top 50 nouns in the data. All these nouns represent potential entities.

5.6. Expected Impact, Conclusions, and Future Work

In our analysis, we proposed an automated approach for classifying and modeling user feedback in a complex and dynamic software ecosystem. Our generated model can provide valuable high-level and ecosystem-wide information to app developers. Using such information, developers can adjust their release engineering strategies to focus on features that enhance the sustainability of their apps and address the shortcomings of poorly executed features. This can be particularly useful for smaller businesses and startups trying to break into the app market [124, 43]. Startups are different from traditional companies in the sense that they have to immediately and accurately identify and implement a product [52], often known as the Minimum Viable Product (MVP), that delivers actual customer value [124, 132]. Our models can serve as a core asset that will help startup companies, operating under significant time and market pressure and with little operating history, to get a quick and comprehensive understanding of the main pressing issues in their ecosystem of operation. Such knowledge can then be utilized to redirect developers' effort toward important user concerns in their app category.

Furthermore, user concerns in our analysis can be extended to cover any non-technical concerns of app users in any app domain (e.g., psychological, business, educational, social, political, etc.). This will facilitate interdisciplinary research between software engineering and other fields of science. In particular, as mobile technology is becoming increasingly ubiquitous, researchers are becoming more interested in understanding how such technology impacts people's lives. For example, Dennison et al. conducted a large-scale qualitative study of health apps to explore users' perspectives and views of features that might support health behavior change [37], Sumter et al. studied the motivations and concerns of adults using online dating apps [151], and Zydney and Warner conducted a study of science learn-

Table 5.6. Associations found in our qualitative analysis (e.g., driver, restaurant, customer service, orders), and the closest corresponding associations from the model.

Found in Data	**Model Association(s)**
Driver associated with ***timing***, particularly regarding inaccurate time ***estimates***	<delivery, estimate>
Complaints regarding ***drivers'*** ability to follow driving directions; unable to find customer's ***location***	<driver, find>
Posts about the friendliness and ***unfriendliness*** of ***drivers***	<driver, awful>
Posts requesting more ***restaurants***; complimenting restaurant ***selection***; complaining about absence of options	<restaurant, show>
Posts discussing ***communication*** between user and ***restaurant***, especially user having to call the restaurant to rectify error	<restaurant, call>
Users requesting a ***refund*** from ***customer service***	<customer, refund>
Complaints about an inability to find ***contact*** information for ***customer service***	N/A
Users discussing calling ***customer service*** to complain about ***inaccurate*** orders	<customer, incorrect>
General complaints about the ***inadequacy*** of ***customer service***	<service, poor> <service, terrible> <customer, poor> <customer, terrible>
Complaints about ***orders*** being ***delayed*** beyond a reasonable amount of ***time***; orders not ***leaving*** the restaurant due to driver tardiness	<order, leave> <time, waste>
Complains about ***orders*** being ***incorrect*** or ***missing*** items.	<customer, incorrect>
Complains about ***order temperature***; orders being delivered cold	<order, hot> <order, cold>

Table 5.7. The most important words in our dataset ranked by TF.IDF (top to bottom, right to left).

order	hour	card	work	help
food	minute	issue	credit	place
service	way	people	min	one
app	fee	worst	promo	everything
delivery	refund	support	dollar	address
time	account	area	half	tip
customer	great	number	day	charge
driver	experience	item	person	bad
restaurant	company	option	problem	thanks
money	code	phone	business	thing

ing apps, their features, and their impact on students' learning outcomes [168]. Generally speaking, the goal of these studies is to understand the complex and increasingly coupled relationships between humans and mobile technology, and to provide practical guidelines for the design of apps that improve people's lives and minimize unwanted side effects. However, due to the lack of automated support, the majority of these studies are conducted via labor-intensive qualitative research methods, such as user surveys and interviews. These methods typically suffer from several limitations, such as the small sample size and the subjectivity and bias of research participants [37, 167]. The proposed models can target these limitations by providing researchers with a framework for large-scale data collection and automated analysis and synthesis. This will enable them to zoom-in into their domains of interest and automatically track specific users' concerns over time, thus formulate more accurate research hypotheses and draw more systematic conclusions.

5.6.1. Threats to Validity

In terms of limitations, the case-study presented in this chapter has several limitations that could potentially limit the validity of the results. For instance, similar to our analysis in Chapter 3, the analysis presented in this chapter takes the form of a case study. In empirical software engineering research, case studies are used to establish in-depth, multi-

faceted exploration of complex issues in their real-life contexts [167]. While case studies can be more realistic and easier to plan than experiments, their results often suffer from external validity threats [167]. One potential threat to our external validity is the dataset used in our analysis; our dataset is limited in size and is generated from a limited number of apps and sources (two app stores and Twitter). To mitigate this threat, we collected our posts (tweets and reviews) over an extended period of time to capture as much information about our ecosystem of interest as possible.

Internal validity threats can stem from the fact that, in our analysis, we only relied on the textual content of the reviews and their sentiment as classification features. In the literature, other types of meta-data attributes, such as the star-rating, author, app version, or submission time of the review, or number of *likes* and *retweets* of Twitter posts, have also been considered as classification features [58, 107]. However the results showed that such attributes had an overall low precision for predicting the type of the review. Nonetheless, we do acknowledge the fact that considering such attributes in the context of review or Twitter data classification might lead to different results. Other internal validity issues might arise from the specific algorithms (SVM and NB) and text processing tools (NLTK) used in our analysis. However, these tools and algorithms have been heavily used in related literature and have been shown to achieve decent performance levels especially in the context of short-text (online reviews and tweets).

A potential threat to the proposed study's internal validity is the fact that subjective human judgment was used to prepare our ground-truth datasets. This includes the manual classification of our dataset. Despite these subjectivity concerns, it is not uncommon in text classification tasks to use humans' judgment to prepare the ground-truth. Therefore, these threats are inevitable; however, they can be partially mitigated by following a systematic classification procedure using multiple judges.

In this chapter, we proposed an automated approach for modeling crowed feedback in app ecosystems. The proposed approach is evaluated through a case study targeting

the ecosystem of the food courier, or delivery, apps. Our results showed that users tend to express a variety of concerns in their feedback. Such concerns can extend over a broad range of issues, such as technical or business. The results also showed that, in our ecosystem of interest, business concerns were easier to be automatically detected and classified. In the third phase of our analysis, we proposed an approach for automatically generating abstract conceptual models of the main use business concerns in the ecosystem of food delivery apps. The results showed that descriptive models can be generated by relying on the frequency and co-occurrence statistics of nouns, verbs, and adjectives in textual user feedback.

In addition to running more case studies over other ecosystems, our future work in this chapter will be aimed at producing more descriptive models. Specifically, our model generation algorithm in this chapter only relied on the frequency and co-occurrence statistics of different nouns, verbs, and adjectives to generate the relationships in our model. However, other variables, such as the priority of user concerns, or the magnitude/direction of the relation between two ecosystem entities, can also be considered to generate more informative models. Another direction of future work will be dedicated to the extrinsic evaluation of our models. Extrinsic evaluation is concerned with criteria relating to the system's function, or role, in relation to its purpose (e.g., validation through experience). To conduct such analysis, our models will be provided to selected groups of app developers to be used as an integral part of their app development activities. Evaluation data will be collected through surveys and interviews along with quantitative data that will measure the level of adaptation as well as the impact of such models on idea formulation and the success or failure of mobile app products.

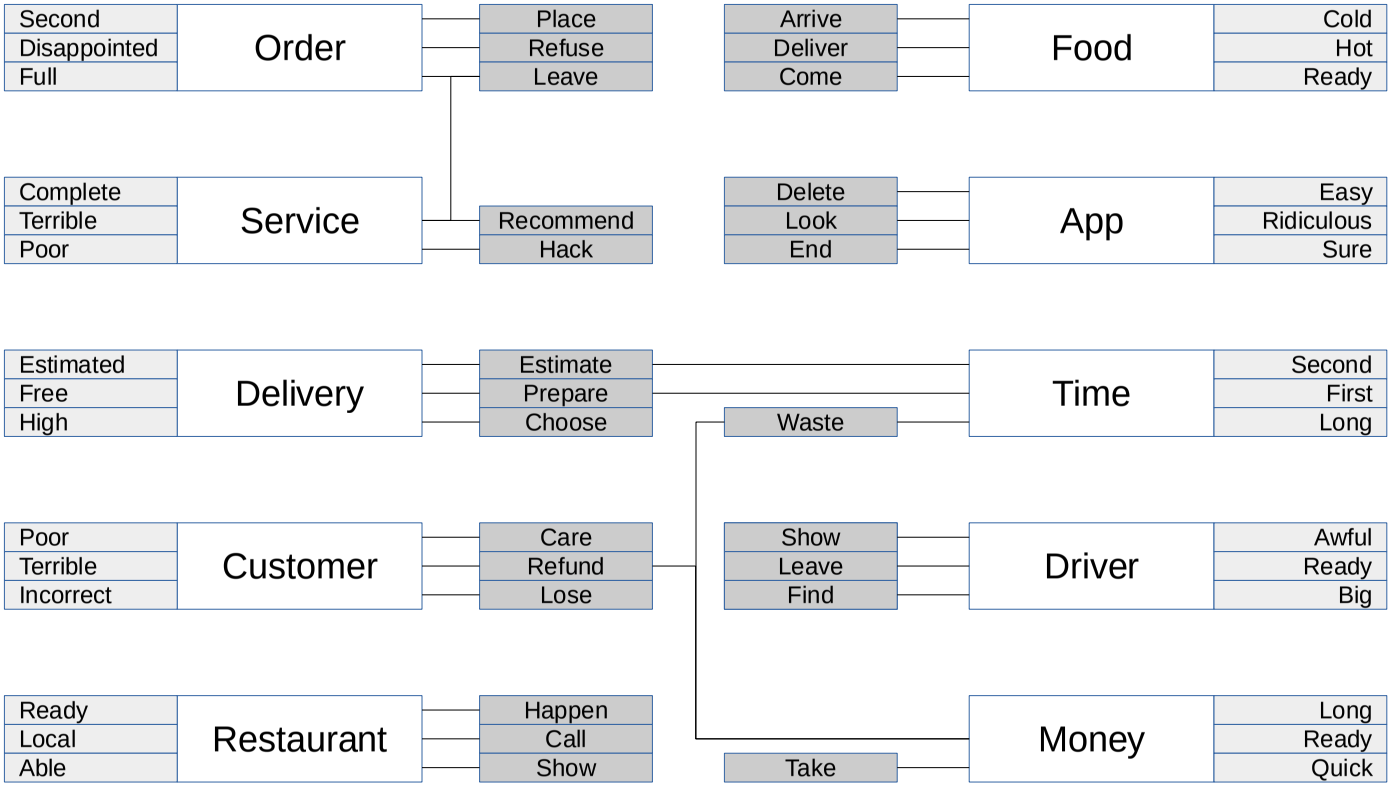

Figure 5.7. A suggested model diagram depicting the relationships between important nouns (entities of the ecosystem), adjectives (attributes), and verbs (functionalities).

Chapter 6. Future Work

Our future work will proceed in several broad directions, aimed at enhancing the applicability, information content, and accuracy of the methods and tools we introduced in this dissertation. These directions include:

- **Social Metric Ranking**: Our proposed approaches for discovering and presenting user-driven requirements mainly utilize the textual information in tweets and reviews. However, some suggestions are more important than others. For example, some bugs or features may be more *severe* than others, indicating a need for further prioritization in order to separate critical errors from minor inconveniences. Existing work on customer interactions on social media has revealed the presence of *influencers*, or important users whose opinions influence the behavior of their peers [17, 82]. In addition, social media platforms, such as Twitter, contain rich metadata that allow posts to be ranked by their visibility and user popularity (i.e., number of likes, favorites, and re-tweets). Therefore, one of our future work direction will be focused on incorporating this information into our models. The main objective is to separate highly popular requests from those that are spurious or unpopular, especially when ranking user feedback by perceived importance. A key challenge underlying this line of work will be to avoid spam posts. Spammers often attempt to fake popularity in order to achieve visibility, obscuring the same metrics that are used to identify influencers [10].

- **Generative Summaries**: Our analysis in Chapter 2 showed that software-relevant tweets could be meaningfully summarized using frequency-based techniques. However, these summaries were *extractive*, drawing entire posts into a ranked list. This limitation prioritizes a particularly important tweet at the expense of other, similar tweets, and therefore could potentially result in missing key phrases and requests. An alternative approach is to create *generative* summaries, which are formed from

users' word distributions in order to create a unique post from scratch [60, 138]. This technique has the potential to produce more representative posts than a purely extractive approach. These posts will essentially represent *idealized* requirements or bug reports. The main challenge underlying this line of work will involve correctly identifying the key entities discussed in tweets and reviews and developing accurate models for generating software-relevant summaries in natural language.

- **Crowdsourcing Source Data**: The approaches presented in Chapters 2, 3, and 5 of this dissertation rely on supervised machine-learning classifiers that are trained on several thousand manually-labeled posts. While our results suggested that identifying technical and business-relevant posts is feasible, current research suggests that state-of-the-art classification, especially when based on deep-learning approaches, requires on the order of tens to hundreds of thousands of training instances [20, 30]. Manually classifying this many posts is impractical, requiring more scalable techniques. One increasingly popular technique involves utilizing crowdsourcing platforms, such as Amazon's Mechanical Turk, to outsource training data labelling to the public [119]. This approach would enable an enormous quantity of data to be collected, thus, enabling new machine learning approaches that require such massive amounts of data. However, opening up training to the crowd also results in additional challenges related to quality control [86]. Specifically, individual users often generate low quality classifications in a bid to increase their output. Solving this problem involves scaling horizontally, potentially having many users judge a single post. The possibility for disagreement among judges implies that the resulting dataset is no longer a ground truth, and our uncertainty about a given judgment must be made part of the model in order to maximize performance.

Works Cited

[1] Sentistrength. http://sentistrength.wlv.ac.uk/. Accessed: August, 15th.

[2] Twitter developer API. https://dev.twitter.com/. Accessed: August, 15th.

[3] Weka 3: Data mining software in java. http://www.cs.waikato.ac.nz/ ml/weka/. Accessed: August, 15th.

[4] App stores - statistics and facts (https://www.statista.com/topics/1729/app-stores/), 2018.

[5] App annie (https://www.appannie.com/en/), 2019.

[6] Statista, market: Online food delivery (https://www.statista.com/outlook/374/109/online-food-delivery/united-states), 2019.

[7] R. Abbott. Program design by informal English descriptions. *Communications of the ACM*, 26(11):882–894, 1983.

[8] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In *International Workshop on Variability Modeling of Software-Intensive Systems*, pages 45–54, 2012.

[9] S. Asur and B. Huberman. Predicting the future with social media. In *International Conference on Web Intelligence and Intelligent Agent Technology*, pages 492–499, 2010.

[10] R. Aswani, A. K. Kar, and P. V. Ilavarasan. Detection of spammers in Twitter marketing: A hybrid approach using social media analytics and bio inspired computing. *Information Systems Frontiers*, 20(3):515–530, 2018.

[11] M. Bano and D. Zowghi. A systematic review on the relationship between user involvement and system success. *Information and Software Technology*, 58:148–169, 2015.

[12] E. Barker, M. Paramita, A. Funk, E. Kurtic, A. Aker, J. Foster, M. Hepple, and R. Gaizauskas. What's the issue here?: Task-based evaluation of reader comment summarization systems. In *International Conference on Language Resources and Evaluation*, pages 23–28, 2016.

[13] A. Bermingham and A. Smeaton. Classifying sentiment in microblogs: Is brevity an advantage? In *International Conference on Information and Knowledge Management*, pages 1833–1836, 2010.

[14] S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[15] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[16] J. Bollen, H. Mao, and X. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8, 2011.

[17] N. Booth and J. A. Matic. Mapping and leveraging influencers in social media to shape corporate brand perceptions. *Corporate Communications*, 16(3):184–191, 2011.

[18] G. Bougie, J. Starke, M. Storey, and D. German. Towards understanding Twitter use in software engineering: Preliminary findings, ongoing challenges and future questions. In *International Workshop on Web 2.0 for Software Engineering*, pages 31–36, 2011.

[19] G. Bouma. Normalized (pointwise) mutual information in collocation extraction. *German Society for Computation Linguistics and Language Technology*, pages 31–40, 2009.

[20] S. Bowman, G. Angeli, C. Potts, and C. Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[21] P. Brandtzæg and J. Heim. Why people use social networking sites. In *International Conference on Online Communities and Social Computing*, pages 143–152, 2009.

[22] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[23] P. Brusilovsky, A. Kobsa, and W. Nejdl, editors. *The Adaptive Web: Methods and Strategies of Web Personalization*. Springer-Verlag, 2007.

[24] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining Knowledge Discovery*, 2(2):121—-167, 1998.

[25] O. Carare. The impact of bestseller rank on demand: Evidence from the app market. *International Economic Review*, 53(3):717–742, 2012.

[26] G. Carreńo and K. Winbladh. Analysis of user comments: An approach for software requirements evolution. In *International Conference on Software Engineering*, pages 343–348, 2013.

[27] N. Chen, J. Lin, S. Hoi, X. Xiao, and B. Zhang. AR-miner: Mining informative reviews for developers from mobile app marketplace. In *International Conference on Software Engineering*, pages 767–778, 2014.

[28] K. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computer Linguistics*, 16(1):22–29, 1990.

[29] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina. Goal-centric traceability for managing non-functional requirements. In *International Conference on Software Engineering*, pages 362–371, 2005.

[30] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *international conference on Machine learning*, pages 160–167, 2008.

[31] M. Conover, B. Goncalves, J. Ratkiewicz, A. Flammini, and F. Mencze. Political polarization on Twitter. In *Conference on Weblogs and Social Media*, pages 89–96, 2011.

[32] M. Conover, B. Goncalves, J. Ratkiewicz, A. Flammini, and F. Mencze. Predicting political preference of Twitter users. In *Advances in Social Networks Analysis and Mining*, pages 289–305, 2013.

[33] P. Coulton and W. Bamford. Experimenting through mobile apps and app stores. *International Journal on Mobile Human Computer Interaction*, 3(4):55–70, 2011.

[34] K. Czarnecki, S. She, and A. Wasowski. Sample spaces and feature models: There and back again. In *International Software Product Line Conference*, pages 22—31, 2008.

[35] A. Damasio. *Descartes' Error: Emotion, Reason, and the Human Brain*. Penguin Books, 2005.

[36] A. Dean and D. Voss. *Design and Analysis of Experiments*. Springer, 1999.

[37] L. Dennison, L. Morrison, G. Conway, and L. Yardley. Opportunities and challenges for smartphone applications in supporting health behavior change: Qualitative study. *Journal of Medical Internet Research*, 15(4):e86, 2013.

[38] P. Dodds, K. Harris, I. Kloumann, C. Bliss, and C. Danforth. Temporal patterns of happiness and information in a global social network: Hedonometrics and Twitter. *PLoS ONE*, 6(12), 2011.

[39] M. Elbendak, P. Vickers, and N. Rossiter. Parsed use case descriptions as a basis for object-oriented class model generation. *Journal of Systems and Software*, 87(7):1209–1223, 2011.

[40] A. Ferrari, G. Spagnolo, and F. Dell'Orletta. Mining commonalities and variabilities from natural language documents. In *International Software Product Line Conference*, pages 116–120, 2013.

[41] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. App store analysis: Mining app stores for relationships between customer, business and technical characteristics. Technical Report rN/14/10, University of College London, 2014.

[42] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Conference on Knowledge Discovery and Data Mining*, pages 1276–1284, 2013.

[43] C. Giardino, X. Wang, and P. Abrahamsson. Why early-stage software startups fail: A behavioral framework. In *International Conference of Software Business*, pages 27–41, 2014.

[44] O. Glassey. Method and instruments for modeling integrated knowledge. *Knowledge and Process Management*, 15(4):247–257, 2008.

[45] M. Glinz. On non-functional requirements. In *Requirements Engineering*, pages 21–26, 2007.

[46] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. In *Stanford Digital Library Technologies Project*, 2009.

[47] M. Gomez, B. Adams, W. Maalej, M. Monperrus, and R. Rouvoy. App store 2.0: From crowdsourced information to actionable feedback in mobile ecosystems. *IEEE Software*, 34(2):81–89, 2017.

[48] M. Gomez, R. Rouvoy, M. Monperrus, and L. Seinturier. A recommender system of buggy app checkers for app store moderators. In *Mobile Software Engineering and Systems*, pages 1–11, 2015.

[49] B. Gonzales-Baixauli, J. C. S. Prado Leite, and J. Mylopoulos. Visual variability analysis for goal models. In *International Requirements Engineering Conference*, pages 198–207, 2004.

[50] J. Gordon. Creating knowledge maps by exploiting dependent relationships. *Knowledge Based Systems*, 13:71–79, 2000.

[51] J. Graham. Yik Yak, the once popular and controversial college messaging app, shuts down, 2017.

[52] C. Gralha, D. Damian, A. Wasserman, M. Goulao, and J. Araújo. The evolution of requirements practices in software startups. In *International Conference on Software Engineering*, 2018.

[53] J. Grimmer and B. Stewart. Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis*, 21:267–297, 2013.

[54] E. Groen, S. Kopczyńska, M. Hauer, T. Krafft, and J. Doerr. Users: The hidden software product quality experts?: A study on how app users report quality aspects in online reviews. In *International Requirements Engineering Conference*, pages 80–89, 2017.

[55] E. Groen, N. Seyff, R. Ali, F. Dalpiaz, J. Doerr, E. Guzman, M. Hosseini, J. Marco, M. Oriol, A. Perini, and M. Stade. The crowd in requirements engineering: The landscape and challenges. *IEEE Software*, 34(2):44–52, 2017.

[56] E. Guzman, R. Alkadhi, and N. Seyff. A needle in a haystack: What do Twitter users say about software? In *International Requirements Engineering Conference*, pages 96–105, 2016.

[57] E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *International Requirements Engineering Conference*, pages 11–20, 2017.

[58] E. Guzman and W. Maalej. How do users like this feature? A fine grained sentiment analysis of app reviews. In *Requirements Engineering*, pages 153–162, 2014.

[59] U. Hahn and I. Mani. The challenges of automatic summarization. *Computer*, 33(11):29–36, 2000.

[60] S. Harabagiu and A. Hickl. Relevance modeling for microblog summarization. In *International Conference on Weblogs and Social Media*, 2011.

[61] N. Hariri, C. Castro-Herrera, M. Mirakhorli, J. Cleland-Huang, and B. Mobasher. Supporting domain analysis through mining and recommending features from online product listings. *Transactions on Software Engineering*, 39(12):1736–1752, 2013.

[62] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: MSR for app stores. In *Conference on Mining Software Repositories*, page 2012, 108–111.

[63] T. Ho. Random decision forests. In *Document Analysis and Recognition*, pages 278–282, 1995.

[64] S. Hoeller. This app lets you text without WiFi or a data plan anywhere in the world — even on a plane, 2016.

[65] J. Holt, S. Perry, and M. Brownsword. *Model-Based Requirements Engineering*. The Institution of Engineering and Technology, 2011.

[66] A. Holzinger, P. Yildirim, M. Geier, and K. Simonic. Quality-based knowledge discovery from medical text on the web. In G. Pasi, G. Bordogna, and L. Jain, editors, *Quality Issues in the Management of Web Information*, pages 145–158. Springer Berlin Heidelberg, 2013.

[67] S. Ihm, W. Loh, and Y. Park. App analytic: A study on correlation analysis of app ranking data. In *International Conference on Cloud and Green Computing*, pages 561–563, 2013.

[68] D. Inouye and J. Kalita. Comparing Twitter summarization algorithms for multiple post summaries. In *International Conference on Social Computing (SocialCom) and International Conference on Privacy, Security, Risk and Trust (PASSAT)*, pages 298–306, 2011.

[69] S. Jansen, A. Finkelstein, and S. Brinkkemper. A sense of community: A research agenda for software ecosystems. In *International Conference on Software Engineering - Companion Volume*, pages 187–190, 2009.

[70] S. Jarzabek, B. Yang, and S. Yoeun. Addressing quality attributes in domain analysis for product lines. *IEEE Software Proceedings*, 153(2):61–73, 2006.

[71] N. Jha and A. Mahmoud. MARC: A mobile application review classifier. In *Requirements Engineering: Foundation for Software Quality: Workshops*, pages 1–15, 2017.

[72] N. Jha and A. Mahmoud. Mining user requirements from application store reviews using frame semantics. In *Requirements Engineering: Foundation for Software Quality*, pages 273–287, 2017.

[73] N. Jha and A. Mahmoud. Using frame semantics for classifying and summarizing application store reviews. *Empirical Software Engineering*, 23(6):3734–3767, 2018.

[74] T. Joachims. Text categorization with Support Vector Machines: Learning with many relevant features. In *European Conference on Machine Learning*, pages 137–142, 1998.

[75] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering*, 22(5):2543–2584, 2017.

[76] R. Junco. Yik Yak and online anonymity are good for college students, 2015.

[77] N. Kaji and M. Kitsuregawa. Building lexicon for sentiment analysis from massive collection of HTML documents. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 1075–1083, 2007.

[78] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institite, Carnegie Mellon University, 1990.

[79] M. Katz. These failed apps discovered a hidden rule of the web, 2017.

[80] E. Khabiri, J. Caverlee, and C. Hsu. Summarizing user-contributed comments. In *Conference on Weblogs and Social Media*, pages 534–537, 2011.

[81] S. Khatiwada, M. Tushev, and A. Mahmoud. Just enough semantics: An information theoretic approach for IR-based software bug localization. *Information and Software Technology*, 93:45–57, 2017.

[82] M. Kunz, B. Hackworth, P. Osborne, and J. High. Fans, friends, and followers: Social media in the retailers' marketing mix. *Journal of Applied Business and Economics*, 12(3):61–68, 2011.

[83] Z. Kurtanović and W. Maalej. Mining user rationale from software reviews. In *International Requirements Engineering Conference*, pages 61–70, 2017.

[84] M. Lane and P. Coleman. Technology ease of use through social networking media. *Journal of Technology Research*, 3:1–12, 2012.

[85] P. Langley, W. Iba, and K. Thompson. An analysis of Bayesian classifiers. In *Artificial Intelligence*, pages 223–228, 1992.

[86] M. Lease. On quality control and machine learning in crowdsourcing. In *Conference on Artificial Intelligence*, 2011.

[87] G. Lee and T. Raghu. Determinants of mobile apps' success: Evidence from the app store market. *Journal of Management Information Systems*, 31(2):133–170, 2014.

[88] K. Lee, D. Palsetia, R. Narayanan, M. Patwary, A. Agrawal, and A. Choudhary. Twitter trending topic classification. In *International Conference on Data Mining Workshops*, pages 251–258, 2011.

[89] D. Levinthal and J. March. A model of adaptive organizational search. *Journal of Economic Behavior and Organization*, 2(4):307–333, 1981.

[90] H. Li, X. Lu, X. Liu, T. Xie, K. Bian, F. Lin, Q. Mei, and F. Feng. Characterizing smartphone usage patterns from millions of android users. In *Conference on Internet Measurement*, pages 459–472, 2015.

[91] X. Lian, J. Cleland-Huang, and L. Zhang. Mining associations between quality concerns and functional requirements. In *International Requirements Engineering Conference*, pages 292–301, 2017.

[92] L. Lim and J. Bentley. How to be a successful app developer: Lessons from the simulation of an app ecosystem. *ACM SIGEVOlution*, 6(1):2–15, 2012.

[93] S. Lim and P. Bentley. Investigating app store ranking algorithms using a simulation of mobile app ecosystems. In *Congress on Evolutionary Computation*, pages 2672–2679, 2013.

[94] S. Lim and A. Finkelstein. Stakerare: using social networks and collaborative filtering for large-scale requirements elicitation. *IEEE transactions on software engineering*, 38(3):707–735, 2012.

[95] S. L. Lim and P. Bentley. Investigating app store ranking algorithms using a simulation of mobile app ecosystems. In *Congress on Evolutionary Computation*, pages 2672–2679, 2013.

[96] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering: How far can we go? In *International Conference on Software Engineering*, pages 94–104, 2018.

[97] C. Lin. Rouge: A package for automatic evaluation of summaries. In *Workshop on Text Summarization Branches Out*, pages 74–81, 2004.

[98] K. Lin and H. Lu. Why people use social networking sites: An empirical study integrating network externalities and motivation theory. *Computers in Human Behavior*, 27(3):1152–1161, 2011.

[99] K. Liu, W. Li, and M. Guo. Emoticon smoothed language models for Twitter sentiment analysis. In *Conference on Artificial Intelligence*, pages 1678–1684, 2012.

[100] C. Llewellyn, C. Grover, and J. Oberlander. Summarizing newspaper comments. In *International Conference on Weblogs and Social Media*, pages 599–602, 2014.

[101] Localytics. Cheat sheet: Overall app benchmarks h2 2016, 2016.

[102] E. Loper, E. Klein, and S. Bird. *Natural language processing with Python*. O'Reilly, 2009.

[103] J. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31, 1968.

[104] M. Lubars, C. Potts, and C. Richter. A review of the state of the practice in requirements modeling. In *International Symposium on Requirements Engineering*, pages 2–17, 1993.

[105] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik. On the automatic classification of app reviews. *Requirements Engineering*, 21(3):311–331, 2016.

[106] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *Requirements Engineering Conference*, pages 116–125, 2015.

[107] W. Maalej, M. Nayebi, T. Johann, and G.Ruhe. Toward data-driven requirements engineering. *IEEE Software*, 33(1):48–54, 2016.

[108] P. Madsen and V. Desai. Failing to learn? The effects of failure and success on organizational learning in the global orbital launch vehicle industry. *Academy of Management Journal*, 53(3):451–476, 2010.

[109] J. Mahler. Who spewed that abuse? Anonymous Yik Yak app isn't telling, 2015.

[110] A. Mahmoud and G. Bradshaw. Estimating semantic relatedness in source code. *ACM Transactions on Software Engineering and Methodology*, 25(1):1–35, 2015.

[111] A. Mahmoud and G. Williams. Detecting, classifying, and tracing non-functional software requirements. *Requirements Engineering*, 21(3):357–381, 2016.

[112] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A survey of app store analysis for software engineering. *Transactions on Software Engineering*, 43(9):817–847, 2017.

[113] A. McCallum and K. Nigam. A comparison of event models for Naive Bayes text classification. In *AAAI - Learning for Text Categorization*, pages 41–48, 1998.

[114] M. McCord and M. Chuah. Spam detection on Twitter using traditional classifiers. In *International conference on Autonomic and trusted computing*, pages 175–186, 2011.

[115] S. Mcilroy, W. Shang, N. Ali, and A. Hassan. User reviews of top mobile apps in apple and google app stores. *Communications of the ACM*, 60(11):62–67, 2017.

[116] T. Mens, M. Claes, P. Grosjean, and A. Serebrenik. *Studying Evolving Software Ecosystems based on Ecological Models*, pages 297–326. Springer Berlin Heidelberg, 2014.

[117] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *National Conference on Artificial Intelligence*, pages 775–780, 2006.

[118] E. Momeni, C. Cardie, and M. Ott. Properties, prediction, and prevalence of useful user-generated comments for descriptive annotation of social media objects. In *Conference on Weblogs and Social Media*, pages 390–399, 2013.

[119] B. Mozafari, P. Sarkar, M. Franklin, M. Jordan, and S. Madden. Scaling up crowdsourcing to very large datasets: a case for active learning. *Proceedings of the VLDB Endowment*, 8(2):125–136, 2014.

[120] J. Mylopoulos, L. Chung, S. Liao, H. Wang, and E. Yu. Exploring alternatives during requirements analysis. *IEEE Software*, 18(1):92–96, 2001.

[121] A. Nenkova and L. Vanderwende. The impact of frequency on summarization. Technical report, Microsoft Research, 2005.

[122] D. Newman, Y. Noh, E. Talley, S. Karimi, and T. Baldwin. Evaluating topic models for digital libraries. In *Annual Joint Conference on Digital Libraries*, pages 215–224, 2010.

[123] A. Nguyen Duc and P. Abrahamsson. Minimum viable product or multiple facet product? The role of MVP in software startups. In *Agile Processes in Software Engineering and Extreme Programming*, pages 118–130, 2016.

[124] A. Nguyen Duc and P. Abrahamsson. Minimum viable product or multiple facet product? The role of mvp in software startups. In *Agile Processes in Software Engineering and Extreme Programming*, pages 118–130, 2016.

[125] J. Nichols, J. Mahmud, and C. Drews. Summarizing sporting events using Twitter. In *International Conference on Intelligent User Interfaces*, pages 189–198, 2012.

[126] F. Nielsen. A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. In *Workshop on 'Making Sense of Microposts': Big things come in small packages*, pages 93–98, 2011.

[127] I. Nonaka. A dynamic theory of organizational knowledge creation. *Organization Science*, 5(1):14–37, 1994.

[128] B. O'Connor, R. Balasubramanyan, B. Routledge, and N. Smith. From tweets to polls: Linking text sentiment to public opinion time series. In *International Conference on Weblogs and Social Media*, pages 122–129, 2010.

[129] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *Requirements Engineering Conference*, pages 125–134, 2013.

[130] A. Pak and P. Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *Language Resources and Evaluation Conference*, pages 1320—1326, 2010.

[131] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall. How can I improve my app? Classifying user reviews for software maintenance and evolution. In *International Conference on Software Maintenance and Evolution*, pages 767–778, 2015.

[132] N. Paternoster, C. Giardino, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson. Software development in Startup companies: A systematic mapping study. *Information and Software Technology*, 56(10):1200–1218, 2014.

[133] T. Petsas, A. Papadogiannakis, M. Polychronakis, E. Markatos, and T. Karagiannis. Rise of the planet of the apps: A systematic study of the mobile app ecosystem. In *Conference on Internet Measurement*, pages 277–290, 2013.

[134] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. 1998.

[135] K. Pohl, G. Boeckle, and F. van der Linden. *Software Product Line Engineering : Foundations, Principles, and Techniques*. Springer, 2005.

[136] P. Prasetyo, D. Lo, P. Achananuparp, Y. Tian, and E. Lim. Automatic classification of software related microblogs. In *International Conference on Software Maintenance*, pages 596–599, 2012.

[137] G. Qiu, B. Liu, J. Bu, and C. Chen. Expanding domain sentiment lexicon through double propagation. In *International Jont Conference on Artificial Intelligence*, pages 1199–1204, 2009.

[138] Z. Ren, S. Liang, E. Meij, and M. de Rijke. Personalized time-aware tweets summarization. In *international conference on Research and development in information retrieval*, pages 513–522, 2013.

[139] A. Sharma, Y. Tian, and D. Lo. Nirmal: Automatic identification of software relevant tweets leveraging language model. In *International Conference on Software Analysis, Evolution, and Reengineering*, page 2015, 449–458.

[140] A. Shontell. The inside story of Yik Yak, 2015.

[141] A. Shontell. Why a girl who was viciously bullied on Yik Yak now believes in the anonymous app's future, 2015.

[142] L. Singer, F. Filho, and M. Storey. Software engineering at the speed of light: How developers stay current using Twitter. In *International Conference on Software Engineering*, pages 211–221, 2014.

[143] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing*, 2013.

[144] A. D. Sorbo, S. Panichella, C. Alexandru, J. Shimagaki, C. Visaggio, G. Canfora, and H. Gall. What would users change in my app? Summarizing app reviews for recommending software changes. In *International Symposium on Foundations of Software Engineering*, pages 499–510, 2016.

[145] D. Sousa, L. Sarmento, and E. Rodrigues. Characterization of the twitter replies network: Are user ties social or topical? In *International Workshop on Search and Mining User-generated Contents*, pages 63–70, 2010.

[146] L. Squires. Enregistering internet language. *Language in Society*, (39):457–492, 2010.

[147] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas. Short text classification in Twitter to improve information filtering. In *Conference on Research and Development in Information Retrieval*, pages 841–842, 2010.

[148] I. Steinwart. On the influence of the kernel on the consistency of support vector machines. *Journal of Machine Learning Research*, 2:67–93, 2001.

[149] S. Stolberg and R. P.-P. a. Wildly popular app Kik offers teenagers, and predators, anonymity, 2016.

[150] J. Storey and E. Barnett. Knowledge management initiatives: Learning from failure. *Journal of Knowledge Management*, 4(2):145–156, 2000.

[151] S. Sumter, L. Vandenbosch, and L. Ligtenberg. Love me Tinder: Untangling emerging adults' motivations for using the dating application tinder. *Telematics and Informatics*, 34(1):67–78, 2017.

[152] Z. Svedic. *The effect of informational signals on mobile apps sales ranks across the globe*. PhD thesis, School Business Faculty, Simon Fraser University, 2015.

[153] M. Thelwall, K. Buckley, and G. Paltoglou. Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, 63(1):163–173, 2012.

[154] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. Sentiment strength detection in short informal text. *Journal of the Association for Information Science and Technology*, 61(12):2544–2558, 2010.

[155] Y. Tian, P. Achananuparp, I. Lubis, D. Lo, and E. Lim. What does software engineering community microblog about? In *Working Conference on Mining Software Repositories*, pages 247–250, 2012.

[156] P. Turney. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In *European Conference on Machine Learning*, pages 491–502, 2001.

[157] P. Turney. Thumbs up or thumbs down?: Semantic orientation applied to unsupervised classification of reviews. In *Annual Meeting on Association for Computational Linguistics*, pages 417–424, 2002.

[158] B. Üstün, W. Melssen, and L. Buydens. Facilitating the application of support vector regression by using a universal pearson vii function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81:29–40, 2006.

[159] D. Vilares, M. Thelwall, and M. Alonso. The megaphone of the people? Spanish SentiStrength for real-time analysis of political tweets. *Journal of Information Science*, 41(6):799–813, 2015.

[160] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *International Conference on Software Engineering*, pages 14–24, 2016.

[161] A. Wang. Don't follow me: Spam detection in Twitter. In *International Conference on Security and Cryptography*, pages 1–10, 2010.

[162] H. Wang and J. Castanon. Sentiment expression via emoticons on social media. In *International Conference on Big Data*, pages 2404–2408, 2015.

[163] S. Wang and C. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Annual Meeting of the Association for Computational Linguistics*, pages 90–94, 2012.

[164] G. Williams and A. Mahmoud. Analyzing, classifying, and interpreting emotions in software users' tweets. In *International Workshop on Emotion Awareness in Software Engineering*, pages 2–7, 2017.

[165] G. Williams and A. Mahmoud. Mining Twitter feeds for software user requirements. In *International Requirements Engineering Conference*, pages 1–10, 2017.

[166] G. Williams and A. Mahmoud. Modeling user concerns in the app store: A case study on the rise and fall of Yik Yak. In *International Requirements Engineering Conference*, pages 64–75, 2018.

[167] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslèn. *Experimentation in Software Engineering*. Springer, 2012.

[168] J. Zydney and Z. Warner. Mobile Apps for science learning: Review of research. *Computers & Education*, 94:1–17, 2016.

Vita

Grant Stuart Williams, born in Port Angeles, Washington, has worked as a software engineer for several years, both at business software development firms and in the process control industry. He received his bachelor's and master's degrees from McNeese State University, Louisiana. In order to broaden his career options and develop his research interests in software requirements mining, he entered the Department of Computer Science at Louisiana State University as a Ph.D. student. Upon completion of his Ph.D., He will be entering the software industry as a data scientist, in the field of requirements engineering.